AUTOMATIC SPEECH RECOGNITION USING DEEP NEURAL NETWORKS: NEW POSSIBILITIES

OSSAMA ABDEL-HAMID

A DISSERTATION SUBMITTED TO
THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

GRADUATE PROGRAM IN
COMPUTER SCIENCE AND ENGINEERING
YORK UNIVERSITY
TORONTO, ONTARIO
September 2014

Abstract

Recently, automatic speech recognition (ASR) systems that use deep neural networks (DNNs) for acoustic modeling have attracted huge research interest. This is due to the recent results that have significantly raised the state of the art performance of ASR systems. This dissertation proposes a number of new methods to improve the state of the art ASR performance by exploiting the power of DNNs.

The first method exploits domain knowledge in designing a special neural network (NN) structure called a convolutional neural network (CNN). This dissertation proposes to use the CNN in a way that applies convolution and pooling operations along frequency to handle frequency variations that commonly happen due to speaker and pronunciation differences in speech signals. Moreover, a new CNN structure called limited weight sharing is proposed to better suit special spectral characteristics of speech signals. Our experimental results have shown that the use of a CNN leads to 6-9% relative reduction in error rate.

The second proposed method deals with speaker variations in a more explicit way through using a new speaker code based adaptation. This method adapts the speech acoustic model to a new speaker by learning a suitable speaker representation based on a small amount of adaptation data from each target speaker. This method alleviates the need to modify any model parameters as is done with other commonly used adaptation methods for neural networks. This greatly reduces the number of parameters to estimate

during adaptation; hence, it allows rapid speaker adaptation.

The third proposed method aims to handle the temporal structure within speech segments by using a deep segmental neural network (DSNN). The DSNN model alleviates the need to use an HMM model as it directly models the posterior probability of the label sequence. Moreover, a segment-aware NN structure has been proposed. It is able to model the dependency among speech frames within each segment and performs better than the conventional frame based DNNs. Experimental results show that the proposed DSNN can significantly improve recognition performance as compared with the conventional frame based models.

Acknowledgments

Foremost, I would like to express my sincere gratitude to my advisor Professor Hui Jiang for the continuous support of my Ph.D. study and research. He allowed me freedom to pursue the research direction I like and in the same time provided me with guidance based on his great knowledge and experience. I am thankful for the time we spent discussing various aspects of my research.

I am thankful to all professors who taught me during my undergraduate study in Cairo University and during my PhD at York University and who helped me grasp different scientific topics.

I would like to express gratitude to York University and the department of computer science for the support and its providing all the needed facilities during the course of my PhD.

I am especially grateful to my wife Mona who accompanied me during the course of my PhD, supported and encouraged me, and filled my days with happiness and joy.

My warm and sincere thanks also go to my friends and colleagues for all the assistance in my research, fruitful discussions and encouragement during my stay at York. I would like to especially mention my friend Abdel-rahman Mohamed for the fruitful deep discussions and for sharing exciting pieces of information in the field of research.

I would like to express gratitude to my parents who planted love of science inside me and the attitude towards excellence through hard work and provided me with encouragement, love, and prayers.

Acronyms

ANN	Artificial neural network
ASR	Automatic speech recognition
BP	Back propagation
CMLLR	Constrained maximum likelihood linear regression
CMVN	Cepstral mean and variance normalization
CNN	Convolutional neural network
CPU	Central processing unit
DBN	Deep belief network
DNN	Deep neural network
DSNN	Deep segmental neural network
EM	Expectation maximization
FFT	Fast Fourier transform
fSA-SC	Feature space speaker adaptation based on speaker code
GMM	Gaussian mixture model
GPU	Graphics processing unit
HMM	Hidden Markov model
LM	Language model
LVASR	Large vocabulary automatic speech recognition
MAP	Maximum a posteriori
MFCC	Mel-frequency cepstral coefficients
MFSC	Mel-frequency spectral coefficients
MLE	Maximum likelihood estimation

MLLR	Maximum likelihood linear regression
MLP	Multi-layer perceptron
MMI	Maximum mutual information
mSA-SC	Model space speaker adaptation based on speaker code
NN	Neural network
PER	Phone error rate
RBM	Restricted Boltzmann machine
RNN	Recurrent neural network
SAT	Speaker adaptive training
SC	Speaker code
VTLN	Vocal tract length normalization
WER	Word error rate

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Automatic speech recognition (ASR) aims to recognize human speech, transforming an audio input into a sequence of words. It is a very challenging task because an ASR system has to convert a variable length speech signal into a variable length sequence of words. Moreover, a successful speaker independent ASR system has to handle variations coming from differences between speakers, in addition to uncertain environmental noises, different speaking styles, and different accents.

Hidden Markov models (HMMs), which model the temporal behavior of speech signals using a sequence of latent states, handle variable-length sequences naturally. Each state is associated with a particular probability distribution of observations. Gaussian mixture models (GMMs) have been, until very recently, regarded as the most powerful model for estimating the probabilistic distribution of speech signals associated with each of these HMM states.

Very recently, HMM models that use artificial neural networks (ANNs) instead of GMMs have witnessed a significant resurgence of research interest. These models have shown significant performance improvement over GMM based models especially when deep neural networks (DNNs) are used. The multiple layers of a DNN allow it to model complex structures of the speech

signals and perform complex processing towards more accurate classification. Moreover, it has been found that the upper layers can learn more invariant representations of the speech signals. Thus, it handles noise and speaker variations in a better way. However, current state of the art systems are still bound to methods and frameworks similar to those that have been developed for GMM-HMM models. DNNs are more powerful and can be used to combine different information sources and predict arbitrary outputs that are not limited to classifications.

1.1 Contributions

This dissertation proposes a number of novel methods that exploit the power of DNNs for improving different aspects of acoustic modeling in ASR.

First of all, the dissertation proposes to use convolutional neural networks (CNNs) to better handle certain variations in the speech signal, improving the speaker invariance of the acoustic model. CNNs have been applied to image analysis tasks to provide invariance to translation and small deformations of images, leading to the state of the art performance on a number of image recognition tasks. This is possible due to the use of convolution and pooling operations. Convolution applies the same filter to different locations. This use of convolution is motivated by translation invariance of certain patterns within different images. Convolution leads to a reduced number of parameters and improved learning of filter weights. The pooling operation reduces the resolution of computed features, and leads to more stability against small deformations of the image.

Similar properties may be important in ASR tasks. Speech signals have certain variations caused by differences in vocal tract length among different speakers. This makes the same speech patterns that result from the same speech units to appear in slightly different frequencies according to the speaker's vocal tract shape. By applying convolution and pooling along the

frequency axis, the resulting neural network enjoys more stability against these variations, leading to better performance. Moreover, the convolution layer neurons receive input from local frequency regions. This results in better stability against band-limited noise. The use of CNNs moves away from a black-box view of neural networks because this design uses domain knowledge to improve the processing of speech signals. The use of CNNs in ASR is described in detail in Chapter 4. Additionally, a novel CNN structure called limited weight sharing that better suits speech signals and leads to better and more efficient modeling is proposed.

Another more explicit method to handle speaker and environment variations is to use speaker and environment adaptation. Chapter 5 presents a novel speaker adaptation method that is based on a learning speaker codes. Speaker adaptation methods aim at optimizing the performance of a speaker independent model towards any target speaker. Traditionally, this is achieved by modifying the model parameters to match the target speaker based on a small amount of adaptation data from the target speaker. Other widely used methods depend on transforming the speech features themselves to match a canonical speaker to which the speech model is optimized. Alternatively, the proposed speaker code based adaptation learns a speaker specific representation that is fed to the DNN along with the speech features without modifying any DNN model parameters. This reduces the number of parameters to learn and allows rapid speaker adaptation that improves the ASR performance with only a very small amount of adaptation data. Additionally, a simple method of scaling features computed by different DNN layers is proposed. It has similar advantages in that it modifies only a relatively small number of parameters. This method can be directly combined with the speaker code based adaptation to achieve further improvement of performance in speaker adaptation.

Chapter 6 focuses on segmental modeling of speech signals. It proposes a deep segmental NN model that is able to model speech variations

within a variable length segment that may represent a sub-word acoustic unit. This method attempts to overcome one of the greatest limitations of frame-based HMM methods: the unrealistic assumption of conditional independence within an HMM state. Moreover, the proposed deep segmental model optimizes NN outputs directly to maximize the posterior probability of the reference label sequence given the whole speech utterance.

1.2 Outline

The first two chapters present background information related to DNNs and ASR. Chapter 2 describes NNs and the advantages of using a deep structure in the NN. Moreover, it describes supervised and unsupervised training algorithms for DNNs. Chapter 3 explains automatic speech recognition (ASR) and the components of an ASR system. It further describes the Hidden Markov Model (HMM) and how it can be used for ASR. Moreover, it surveys different NN based ASR models and how an NN can be combined with an HMM to achieve state of the art ASR performance. The next three chapters describe the three proposed methods to improve the performance of ASR through some innovative uses of NNs. Chapter 4 describes CNN models and how they differ from standard fully connected NNs. Moreover, it proposes using CNNs for acoustic modeling of speech signals and describes the proposed limited weight sharing. Chapter 5 describes the proposed speaker adaptation methods: speaker code based adaptation and a simple speaker adaptive scaling of the DNN activations. Chapter 6 presents the proposed deep segmental neural network model, as well as how it can be learned and used for ASR. Finally, chapter 7 draws all conclusions of this dissertation and discusses possible future work.

Chapter 2

Deep Neural Networks

2.1 Artificial Neural Networks

The ideas of neural networks (NNs) originated from attempts to understand the biological information processing systems in the late 19th century. The concept of a neural network was possibly first published in 1943 in [1]. However, artifical neural networks were not widely used until the 1980s, following publication of the back-propagation algorithm for training a multi-layered feed-forward ANN [2].

Although they are inspired by biological neural systems, ANN models do not necessarily mimic the workings of biological networks, which are still not fully understood. ANNs have many architectures and training algorithms. An ANN has a number of neuron-like nodes (called artificial neurons or simply neurons) that process information coming to them from other interconnected nodes. Usually the connections have adaptive weights to be estimated during learning.

ANN models have shown success in many diverse applications, including

- object classification like face and character recognition,
- function approximation and regression, and

- data processing such as clustering and filtering.

One of the most successful examples of an ANN is the *feed-forward NN* which is also called a *multi-layer perceptron* (MLP). It can be used to perform classification or regression. The network usually has an input layer, one or more intermediate hidden layers, and an output layer. The network neurons have an activation function that defines the neuron output as a function of the input coming from the neuron connections that transfer (and modify) the output of other connected neurons in the lower layer(s). A network of one hidden layer that has a finite (but big enough) number of neurons with a sigmoid activation function is proved by George Cybenko to be a universal approximator in [3]. Moreover, other forms of activation function are proved to be universal approximators as in [4].

Learning of a multi-layer feed-forward NN is usually done using the back-propagation algorithm where the error from the target output is back-propagated through the hidden layers to update the connection weights in all layers simultaneously. But having one or more hidden layers makes the learning objective function non-convex, which implies the risk of being stuck in a bad local minimum during model learning. Another problem is over-fitting which means that the network generalizes badly on other samples not seen in the training set. This problem happens when a network that has a relatively large number of free parameters is trained using a small training set. To solve this problem, either the network size is reduced or a regularization term is added to the objective function to reduce its complexity. Another possible solution is early stopping, where the learning is stopped earlier during training when the network performance decreases on a held-out validation data set. However, these solutions may lead to non-optimal values of the weights.

In addition to being a universal function approximator, a NN can be used to model data distributions as a generative model. A deep generative multi-layer NN that has stochastic binary neurons (given it has enough depth and width) is proven to be a universal distribution approximator for binary multi-

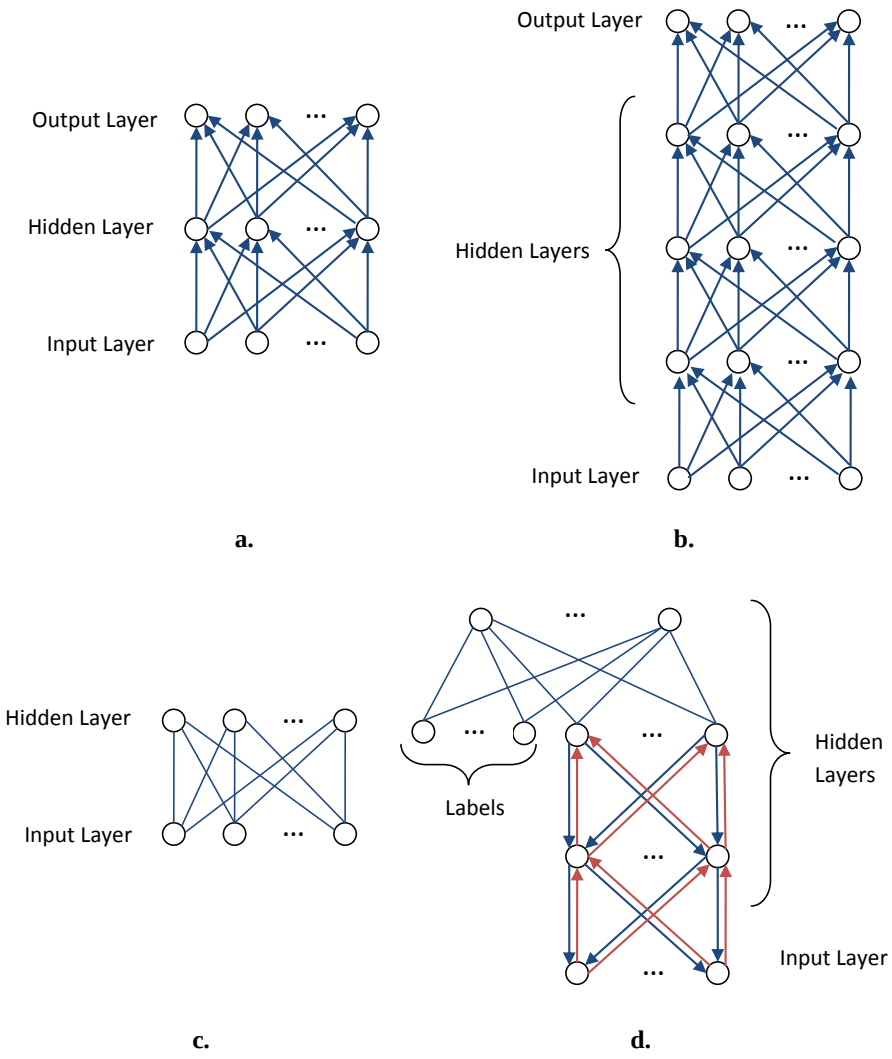


Figure 2.1: Examples of neural network models. a. A standard shallow feed-forward NN. b. A deep feed-forward NN. c. A restricted Boltzmann machine which has undirected connections. d. Deep belief network that has a top layer with undirected weights (an associative memory) that sends signals down through the generative directed weights.

variate data in [5] and [6]. This form of NN is called a *deep belief network* which is described in more detail in section 2.4.4.

Fig.2.1 shows a number of examples of NN models. These models include discriminative feed forward NNs shown in (a) and (b), and generative NNs such as the *restricted Boltzmann machine* (RBM) in (c), and deep belief network in (d). More details about these models are given in the next sections.

2.2 Feed Forward Neural Networks

2.2.1 Formulation

A feed forward neural network has one or more hidden layers in addition to the output layer. Each layer has a number of units (or neurons), each of which takes outputs of the lower layer as input, multiplies them by a weight vector, sums the result and passes it through a non-linear activation function such as *sigmoid* or *tanh*. The output of a unit of a fully connected layer is computed as follows:

$$o_i^{(l)} = \sigma(\sum_j o_j^{(l-1)} w_{ji}^{(l)} + w_{0i}^{(l)}) \tag{2.1}$$

where $o_i^{(l)}$ denotes output of i-th unit in l-th layer, $w_{ji}^{(l)}$ denotes the connecting weight from j-th unit in the layer $l-1$ to i-th unit in the l-th layer, $w_{0i}^{(l)}$ is a bias added to the i-th unit, and $\sigma(x)$ is a non-linear activation function. A sigmoid function is a common choice of the activation function which is defined as follows:

$$\sigma(x) = 1/(1+\exp(-x)) \tag{2.2}$$

In this work, a sigmoid function is assumed by default unless mentioned otherwise.

For simplicity of notation, the above computation can be represented in

vector form as follows:

$$o_i^{(l)} = \sigma(\mathbf{o}^{(l-1)} \cdot \mathbf{w}_i^{(l)}) \tag{2.3}$$

where the bias term is absorbed in the column weight vector $\mathbf{w}_i^{(l)}$ by expanding the vector $\mathbf{o}^{(l-1)}$ with an extra dimension of 1. Furthermore, all neuron activities in each layer can be represented as the following matrix form:

$$\mathbf{o^{(l)}} = \sigma(\mathbf{o}^{(l-1)}\mathbf{W}^{(l)}) \qquad (l = 1, 2, \cdots, L-1) \tag{2.4}$$

where $\mathbf{W}^{(l)}$ denotes weight matrix in the l-th layer and the ith column is the weight vector $\mathbf{w}_i^{(l)}$ for all i.

The first (bottom) layer is the input layer and the topmost layer is the output layer. For a multi-class classification problem, the posterior probability of each class can be estimated using the output softmax layer as:

$$y_i = \frac{\exp\left(o_i^{(L)}\right)}{\sum_j \exp\left(o_j^{(L)}\right)} \tag{2.5}$$

where $o_i^{(L)}$ is computed as $o_i^{(L)} = \mathbf{o}^{(L-1)} \cdot \mathbf{w}_i^{(L)}$. Another option for the output layer is to use sigmoid or linear activation functions.

2.2.2 Learning

To learn the weights of a feed forward NN, a labeled training data set is needed. The aim is to minimize the errors in the NN output given the training inputs. At the same time, the NN predictions should be general, that is able to correctly predict the output for unseen samples. A number of objective functions can be used. One that is widely used is the mean squared error. Another more successful objective function for classification problems (when a softmax output layer is used) is the cross entropy which is defined

as follows:

$$Q(\{\mathbf{W}^{(l)}\}) = -\sum_k \sum_i d_i^{(k)} \log y_i^{(k)}, \tag{2.6}$$

where y_i^k represents the NN ith output given the features of the kth sample and $d_i^{(k)}$ is the target output. The weights are chosen such that the total cross entropy over the training data set is minimized. The cross entropy decreases when the discrepancy between the reference target $\mathbf{d}$ and the softmax layer prediction $\mathbf{y}$ decreases.

The derivative of Q with respect to each weight matrix, $\mathbf{W}^{(l)}$, can be efficiently computed based on the well-known error back-propagation algorithm. If we use the stochastic gradient descent algorithm to minimize the objective function, for each training sample or mini-batch, each weight matrix update can be computed as:

$$\Delta \mathbf{W}^{(l)} = \epsilon \cdot \left(\mathbf{o}^{(l-1)}\right)' \mathbf{e}^{(l)} \qquad (l = 1, 2, \cdots, L) \tag{2.7}$$

where ϵ is learning rate and the error signal vector in l-th layer, $\mathbf{e}^{(l)}$, is computed backwards for the sigmoid hidden unit as follows:

$$\mathbf{e}^{(L)} = \mathbf{d} - \mathbf{y} \tag{2.8}$$

$$\mathbf{e}^{(l)} = \left(\mathbf{e}^{(l+1)} \left(\mathbf{W}^{(l+1)}\right)'\right) \bullet \mathbf{o}^{(l)} \bullet \left(\mathbf{1} - \mathbf{o}^{(l)}\right) \qquad (l = L-1, \cdots, 2, 1) \tag{2.9}$$

where $\bullet$ represents element-wise multiplication of two equally sized matrices or vectors.

2.3 Going Deep

Although a shallow feed-forward NN with two layers (one hidden layer and one output layer) is proved to be a universal function approximator, it may not be efficient in terms of representation or computational complexity [7]. A *deep neural network* (DNN) having more than two layers can provide a more compact method to process inputs. For example, suppose an NN is to be trained to classify objects from images. Lower layers can work to extract low level features like edges. Intermediate layers can combine these edges into more complex shapes. Then top layers do the final classification. Therefore, higher levels of abstraction are generated in the higher layers that may exploit the low level processing done in the lower layers. A shallow architecture may do the same classification but may need to repeat doing the same computations many times in one layer instead of doing it once then exploiting it in higher layers.

Parity computation is demonstrated in [8] as an example that motivates deep architectures. The following are three ways to construct circuits that compute the N-bit boolean parity function:

1. N chained XOR gates which can be thought of as having N layers with each layer having size 1.

2. $N-1$ XOR gates arranged in a tree that has $\log_2 N$ layers and each layer has $\log_2 l$ gates.

3. a DNF formula (disjunctive normal terms i.e. AND minterms in the first layer then an OR function of the results) that has $O(2^N)$ minterms.

It can be seen from this example that although a shallow architecture can do the job, it needs an exponential number of components, while a deep architecture needs only $N-1$ components. A similar argument applies to other problems in pattern classification and recognition. If these components

are to be trained, more training examples will be needed to train the higher number of components.

However, training a DNN is a difficult problem. For example, empirical results suggest that training a DNN using back-propagation is more difficult than training a shallow network [9, 10]. The objective function will be highly non-convex and it will be easy to get stuck in a bad local minimum. In some earlier cases, learning a deeper NN may have yielded worse results than a shallow one [11]. The work in [12] studies why it is more difficult to train a deep NN. An important reason is that many hidden and output units are saturated from the beginning and change slowly with training especially when a sigmoid non-linearity is used. Careful initialization of the weights and choice of the activation function can help in training these deep models.

In spite of this difficulty, there have been some successful examples of learning DNN architectures by stochastic gradient descent. For example, a deep convolutional NN [13] (to be described in chapter 4) can be learned easily using back-propagation. One of the factors that makes convolutional NNs easier to train is the use of weight sharing which reduces the number of free parameters to be learned. The use of weight sharing is based on the translation invariance property of images. The architecture of a convolutional NN makes the network learn low level features that process smaller areas of the image in more details in the low layers, while nodes in higher layers process larger areas with less details resulting in more abstract features. Moreover, the use of pooling layers provides more robustness to small deformations of image parts.

Another successful example of DNNs trained using back-propagation is shown in [14] where a deep multi-layer feed-forward NN could be trained on an OCR dataset by continuously applying automatic elastic deformations to the training set. Therefore, it can be thought of as training the network on an infinite training set which may get rid of the problem of over-fitting. Moreover in this case the objective function is changing in each training iteration that

includes a number of training samples. This can help in escaping from bad local minima.

2.4 Unsupervised Learning

Previous studies showed difficulty in training deep NNs directly with back-propagation [10, 11]. This difficulty arises from the fact that the back-propagation signal gets weaker (in some other cases the signal explodes, for example when linear activation functions are used) as it goes back towards the lower layers, leaving the bottom layers badly trained. This weakened the interest in studying deep models.

In 2006, Hinton et al. [15] showed that a deep NN model can be trained easily by pre-training the model using unsupervised learning. This pre-training was done by using the weights of a deep belief network created by stacking multiple RBMs as described next. After that, more interest has grown on building deep models and supervised and unsupervised training of them. A number of empirical studies have shown that unsupervised pre-training of feed-forward NNs (both deep and shallow) improves their performance [10, 9, 16].

Unsupervised learning of NNs can benefit from unlabeled data in terms of improving the overall classification performance. Moreover, it can improve the supervised learning signals when the generative and discriminative learning signals are combined in a semi-supervised setting [17].

There are a number of other applications of unsupervised learning. Feature learning is an important one among many others. [18] shows that stacked RBMs can learn more expressive and invariant features that improve the classification performance of a support vector machine. Moreover, generative models trained using unsupervised learning can model the probability distribution of the training data. This can be used to synthesize random samples, correct noisy samples, and to complete missing data.

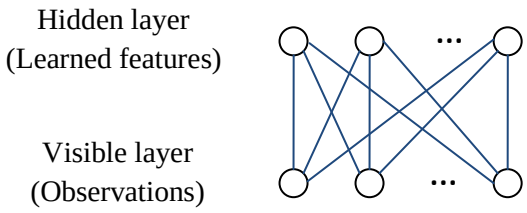


Figure 2.2: RBM

2.4.1 Restricted Boltzmann Machine (RBM)

A *restricted Boltzmann machine* (RBM) is an undirected graphical model organized as a bipartite graph with undirected connections [15, 19]. The lower layer has visible units that represent the raw data (observations), and the higher layer has hidden units that represent hidden features (or latent variables) as shown in figure 2.2. The model is restricted in the sense that it has weighted connections only between visible and hidden units and no connections among visible units nor among hidden units. In the simplest form, the model has binary units in both the visible and hidden layers. The model defines a probabilistic distribution of the joint states of visible and hidden units using an energy function defined using the weights and biases as follows:

$$E(\mathbf{v}, \mathbf{h}|\theta) = -\sum_{i=1}^{V}\sum_{j=1}^{H} w_{ij} v_i h_j - \sum_{i=1}^{V} b_i v_i - \sum_{j=1}^{H} a_j h_j \tag{2.10}$$

where $\mathbf{v} = [v_1 ... v_V]^\top$ represents the states of the visible units, $\mathbf{h} = [h_1 ... h_H]^\top$ represents the states of the hidden units, w_{ij} is the symmetric weight between the ith visible unit and the jth hidden unit, a_i is the bias term of the ith visible unit, and b_j is the bias of the jth hidden unit. θ represents the combined set of parameters $\mathbf{W}$, $\mathbf{b}$, and $\mathbf{a}$.

The probability of the joint states of $\mathbf{v}$ and $\mathbf{h}$ is:

$$p(\mathbf{v}, \mathbf{h}|\theta) = \frac{e^{-E(\mathbf{v},\mathbf{h}|\theta)}}{Z(\theta)} \tag{2.11}$$

where $Z(\theta)$ is the partition function which equals the sum of energies of all states:

$$Z(\theta) = \sum_{\mathbf{v}} \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h}|\theta)} \tag{2.12}$$

The last equation shows that computing the partition function is intractable because the number of terms to sum grows exponentially as the number of the model elements. Fortunately, many practical applications do not need the estimation of the partition function. Given the state of the visible units of the RBM, the hidden units become independent and a factorial probability distribution can be computed easily as follows:

$$p(h_j = 1|\mathbf{v}, \theta) = \sigma(\sum_{i=1}^{V} w_{ij} v_i + a_i) \tag{2.13}$$

where $\sigma(x) = \dfrac{1}{1 + e^{-x}}$. Similarly, given the hidden states, the probability distribution over the visible units is:

$$p(v_i = 1|\mathbf{h}, \theta) = \sigma(\sum_{j=1}^{H} w_{ij} h_j + b_j) \tag{2.14}$$

Computing the probability distribution of visibles can be done by marginalizing over $\mathbf{h}$ in Eq. 2.11 as follows:

$$p(\mathbf{v}|\theta) = \frac{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h}|\theta)}}{\sum_{\hat{\mathbf{v}}} \sum_{\mathbf{h}} e^{-E(\hat{\mathbf{v}},\mathbf{h}|\theta)}} \tag{2.15}$$

Since the above summation is not tractable, it is not practical to use the last equation to generate random samples of the visibles. Instead, Monte Carlo

sampling methods can be used with the conditional probability equations. Good samples can be generated using prolonged iterations of block Gibbs sampling. This can be done starting from random states, then alternating the application of the conditional equations 2.13 and 2.14 to update the states of the visible and hidden units.

2.4.2 RBM Training

A natural way to learn the RBM parameters would be to maximize the log likelihood of data. Suppose having a data sample $\mathbf{v}$, the log likelihood of $\mathbf{v}$ is:

$$\log p(\mathbf{v}|\theta) = \log \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h}|\theta)} - \log \sum_{\hat{\mathbf{v}}} \sum_{\mathbf{h}} e^{-E(\hat{\mathbf{v}},\mathbf{h}|\theta)} \tag{2.16}$$

Unfortunately there is no known analytical solution that can find the model parameters that maximize the above log likelihood function. Instead, a gradient descent optimization can be used by following the first derivative of the log likelihood function. Differentiating Eq. 2.16 with respect to all weights w_{ij} gives:

$$\Delta w_{ij} \propto \langle v_i h_j \rangle_{\text{data}} - \langle v_i h_j \rangle_{\text{model}} \tag{2.17}$$

Where $\langle x \rangle$ is the expectation of the variable x. The first term is the expectation that v_i and h_j are both 1 in the training data, which can be computed easily as the average of the value of visible elements of each data sample multiplied by the probability of hiddens taking the value of 1 given that sample. The second term is the expected value of the model generating samples having $v_i = 1$ and $h_j = 1$. This is not tractable to compute accurately as it requires the averaging over infinite number of samples generated using Gibbs sampling. Instead, the *contrastive divergence* (CD) algorithm [20, 15] can be used to compute an approximate update value by applying one or a limited number of iterations of Gibbs sampling to compute an approximate average. In this way, for each data sample one or more iterations of Gibbs sampling

iterations are applied to get new states of the visible units that are closer to the *expected* value of the visible units according to the model distribution.

Contrastive divergence gives a biased estimate of the objective function. Other training algorithms that better approximate the maximum log likelihood objective function have been proposed. For example, *Persistent Contrastive Divergence* (PCD) has been shown to improve the likelihood estimation[21, 22]. Also, *Particle Filtered MCMC-MLE* has been shown to improve over CD [23].

2.4.3 RBM with continuous valued visible data

For continuous valued data, a binary RBM is not suitable. A simple heuristic can be used by normalizing data between 0 and 1 and considering the data values to be visible units activation probabilities as done in [15]. A better solution is to use a continuous data probability distribution like the Gaussian distribution [24, 19, 10, 25]. In this case, the energy function is defined as:

$$E(\mathbf{v}, \mathbf{h}|\theta) = -\sum_{i=1}^{V}\sum_{j=1}^{H} w_{ij} v_i h_j - \sum_{j=1}^{H} a_j h_j + \frac{1}{2}\sum_{i=1}^{V}(v_i - b_i)^2 \tag{2.18}$$

In this case, the conditional probability of visible states given the hidden states follows a Gaussian density function as:

$$p(v_i|\mathbf{h}, \theta) = \mathcal{N}\left(\sum_{j=1}^{H} w_{ij} h_j + b_i, 1\right) \tag{2.19}$$

where $\mathcal{N}(\mu, \sigma)$ denotes a Gaussian distribution with a mean μ and a standard deviation σ.

The training procedure is the same except that during Gibbs sampling of visible units the samples are drawn according to Eq. 2.19 instead of equation 2.14. Usually with these models, a fixed variance of one is used, which is not optimal. Moreover, modeling the conditional distribution of the visible units

as independent factors may lead to noisy samples and may not be suitable to model low level structures of data like images where pixel values are highly correlated.

A simple modification to solve this dependency problem is to include lateral connections within the visible layer to create a *semi-restricted Boltzmann Machine* (SRBM) [26]. More sophisticated variants are described next, which model both the mean and covariance in a better way.

Factored Three Way RBM

The *three way RBM* [27] includes weights to specify the relations between three units at the same time instead of two (one visible and one hidden units) as the regular RBM. The model energy function is defined as:

$$E(\mathbf{x}, \mathbf{y}, \mathbf{h}) = -\sum_{i,j,k} x_i y_j h_k w_{ijk} \tag{2.20}$$

where $\mathbf{x}$ and $\mathbf{y}$ are two observation vectors whose interactions are to be modeled along with the hidden units $\mathbf{h}$. The biases in the previous equation are omitted for simplicity.

The second observation vector can represent a previous observation where the model can be used to model the conditional relation between consecutive observations as in [27]. It can also be used to model a symbolic state of the model. For example, [28] used it to model the relation between the observation and a style label for modeling different motion styles.

Moreover, the second observation vector may be identical to the first to better model the dependency between different components of the observation. This model is shown in [29] to be able to better model natural images by making each weight specify the interaction between a pixel, a hidden unit, and another pixel at the same time. This will enable it to model the complex structure of images and the dependency between pixels.

The drawback of the definition in Eq. 2.20 is that the model will have

too many weights to learn. A better idea is to factor these weights as a sum of factors, each of which is a three way outer product, so that $w_{ijk} = \sum_f b_{if} c_{jf} p_{kf}$ where b_{if}, c_{jf}, and p_{kf} are the elements of the weight matrices B, C, and P corresponding to the interaction of the three units with factor f, creating a *factored three way RBM* [30, 29]. In this case, Eq. 2.20 is approximated by:

$$E(\mathbf{x}, \mathbf{y}, \mathbf{h}) = -\sum_f \left(\sum_i x_i b_{if}\right)\left(\sum_j y_j c_{jf}\right)\left(\sum_k h_k p_{kf}\right) \tag{2.21}$$

Learning of the model parameters can be done in a similar way to the standard RBM by using gradient ascent to maximize the log likelihood. In the case of modeling the conditional relationship of $\mathbf{y}$ given $\mathbf{x}$ the objective function is the conditional log likelihood: $L = \sum_\alpha \log p(\mathbf{y}^\alpha | \mathbf{x}^\alpha)$. To update a weight w_{if} which can represent any of the three sets of the parameters B, C, or P the following update rule can be used:

$$\Delta w_{if} \propto \langle \frac{\partial E(\mathbf{y}, \mathbf{h}; \mathbf{x})}{\partial w_{if}} \rangle_{\text{data}} - \langle \frac{\partial E(\mathbf{y}, \mathbf{h}; \mathbf{x})}{\partial w_{if}} \rangle_{\text{model}} \tag{2.22}$$

The second term is not tractable but the contrastive divergence algorithm can be used to approximate it. It should be noted that only $\mathbf{y}$ and $\mathbf{h}$ need to be sampled using Gibbs sampling as $\mathbf{x}$ is always visible. In this case the components of $\mathbf{y}$ or $\mathbf{h}$ would be factorial given the other two vectors and sampling would be easy.

Learning in the non-conditional case is more challenging because we will need to get samples of observations given the hidden states. In this case, the observation components are not factorial. But since contrastive divergence training is used, the samples of observations need not to be accurate. They just need to be closer to the joint distribution of the visibles given the hiddens. For the binary case, a mean field approximation can be used [31]. For the continuous case, a covariance matrix of the Gaussian visibles based on

the model weights can be computed easily and inverted to generate a sample given the hiddens but this would be time consuming. More efficiently, an approximate sample can be generated using the hybrid Monte Carlo algorithm [32] as done in [29].

Mean-Covariance RBM

In a factored three way RBM, the states of the hidden units define the covariance of the visible units when two sets of the same visible units are used. To better model both the mean and covariance of a Gaussian model using the hidden units, a *mean-covariance RBM (mcRBM)* can be used, where two sets of hidden units are used. One set defines the means using a standard RBM and the other set defines the covariance using a factored three way RBM [33]. For training, contrastive divergence is used, but it is more challenging since the visible units are not factorial given the hidden units. Hence, hybrid Monte Carlo sampling is used to get random samples of the visible units given the hidden ones.

A similar model where the Gaussian distribution is replaced with a Student's t distribution (which is a better distribution for modeling images) is proposed by Ranzato et al. [34] and it is called *mPoT*.

Spike and Slab RBM

The spike and slab RBM (ssRBM) improves over the standard RBM with Gaussian visible units by including two sets of hidden units: real valued *slab* units and binary valued *spike* units [35]. It has the advantage of enabling the spike units to define a Gaussian distribution covariance matrix over the visible units. At the same time, its energy function enables easy Gibbs block sampling and hence simpler contrastive divergence training than mcRBM which requires the use of slower sampling techniques like hybrid Monte Carlo. It is shown in [36] that this model can learn better features than the mcRBM and other RBM variants.

2.4.4 Deep Belief Network

The simplest method to learn a deep model is to stack separately trained layers where the hidden representations of one layer are considered as the input of the next layer. The intuition behind this method is simple: since the simple models presented in section 2.4 can learn good representations of raw data, these representations can act as raw data for another model to generate even better representations using the more complex transformation of the combined two layers. Theoretical and empirical results show that this method works well.

A deep belief network (DBN) is a good example of this method [15]. It's a generative model that learns a probabilistic distribution of the training data. It consists of a number of layers with directed top-down weights and one top layer having undirected weights. The states of a layer in DBN other than the top two layers follows a factorial probability distribution that depends on the states of the layer above and the top down weights between the two layers as follows:

$$p(h_i^{(l)}|\mathbf{h}^{(l+1)}) = \sigma(\sum_j h_j^{(l+1)} w_{ji}^{(l+1)} + b_i^{(l+1)}) \tag{2.23}$$

where $\mathbf{h}^{(l)}$ represents the state of the lth layer, $w_{ij}^{(l+1)}$ is the weight connecting the jth node of the $l+1$th layer and the ith node of the lth layer. The undirected weights between the top two layers of a DBN constitutes an RBM. To generate a sample from the DBN, first a large enough number of Gibbs sampling iterations are performed to get a sample of the state of the top two layers, then an ancestral top down pass is done to generate samples of the lower layers given the state of the layer above using Eq. 2.23.

Learning of a DBN is done as follows. First learn an RBM using the training data. Use the RBM weights as the weights of the first layer and use them in both bottom-up and top-down directions to generate samples of the hidden units and to reconstruct visible units states given the hidden

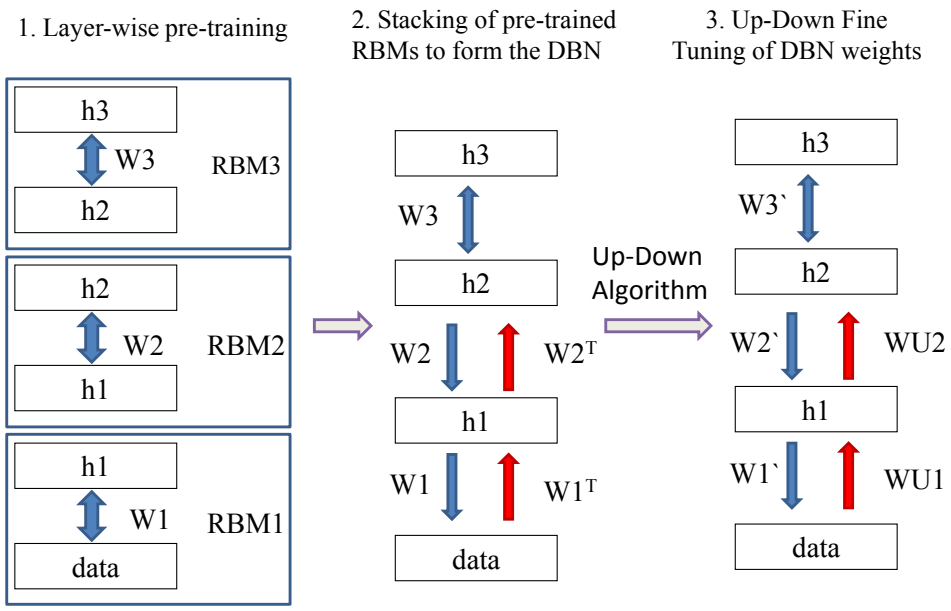


Figure 2.3: Training of DBN. The figure shows the training of a DBN that has three hidden layers. The top two layers have undirected weights in-between. The other weights are directed. The blue arrows represent the weights of the generative model. The red arrows represent inference weights used to approximate the hidden states posteriors efficiently.

states. Use the first layer weights to generate samples of the hidden layer and consider these samples as training data of a second RBM. Repeat this process until all the layers of the DBN are learned as shown in figure 2.3.

Hinton et al. proved in [15] that adding a hidden layer is guaranteed to maximize a lower bound on the likelihood. This happens because as we add more hidden layers a better prior of the hidden layers can be learned. However, this greedy training is not optimal because it does not optimize all the layers together. Fine tuning of the DBN as a whole can bring more gain. The DBN can be initialized using this greedy layer-wise training then the *Up-Down* algorithm [15] can be used afterwards to fine tune all of the layers together to have a better model of the data. The Up-Down algorithm is a modified version of the Wake-Sleep algorithm that has been proposed to train the Helmholtz machine [37]. The wake-sleep algorithm suffers from slow convergence and the mode-averaging problem. With the use of pre-training the convergence of fine tuning is much faster and the use of a small number of Gibbs sampling iterations for the top-down pass makes the model to pick one mode, hence solving the mode averaging problem.

If a DBN is to be used for classification, label units can be added to the units of the layer below the topmost so that the relation between these labels and the features computed at this layer can be captured by the top RBM. Then an estimate of the conditional probability of the label given a data vector can be either computed using sampling techniques or approximated using mean-field estimations. Otherwise, the DBN weights can be used as a pre-trained feed-forward NN that can be fine tuned using the back-propagation algorithm by adding a top softmax layer for labels as described later.

Lee et al. [18] proposed another method of stacking pre-trained RBMs by keeping the connections between stacked layers undirected. This makes generating samples from the model more difficult, though, as each hidden layer can receive inputs from the layers above and beneath it. As a result, a large number of Gibbs sampling iterations are needed to get unbiased samples

from the model or even to compute activations of the topmost hidden layer given a data vector. Otherwise, a mean-field approximation can be used to compute the hidden activations using a relatively small number of iterations. It is empirically shown in [18] that this method can learn complex generative models and generate better features to achieve better classification accuracy.

2.4.5 Pre-training of feed-forward NNs

A feed-forward NN can be trained using the back-propagation algorithm. The back-propagation algorithm depends on the existence of target outputs or labels used to compute the error in the NN output. This error is back-propagated through the network layers and the weights are updated in order to minimize this error.

The back-propagation algorithm can suffer from over-fitting when a small number of training examples are available or when the NN has too many parameters. Moreover, there are other cases when there are many unlabeled training examples and a small number of labeled examples where semi-supervised training can use the unlabeled examples to get a better model by understanding the structure inherent in the data.

Pre-training of the NN can be done by stacking models trained in an unsupervised fashion as described in section 2.4.4. Then a softmax layer can be added for classification and the model is fine tuned to minimize the classification error or the cross entropy using the standard back-propagation algorithm. Actually, this is the most common application of unsupervised learning methods. Many publications have empirically shown the benefit of unsupervised pre-training in training DNNs [15, 10, 16, 38, 39].

Erhan et al. have done extensive experiments to study the gain obtained with using unsupervised pre-training [40]. They concluded the following:

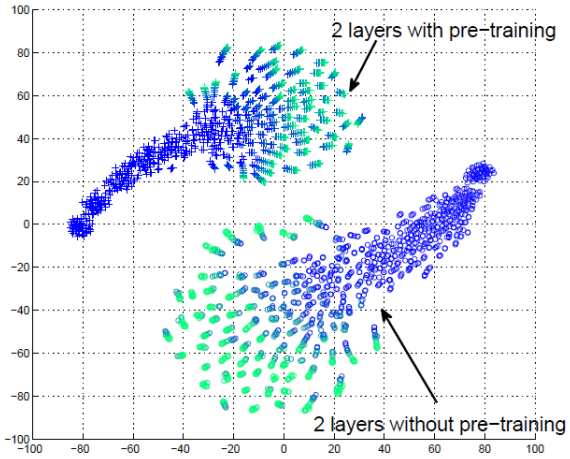
- There's a consistent improvement with unsupervised pre-training.
- The pre-training has a regularization effect. It can achieve better re-

sults with large networks but it can slightly harm the performance when there is a small number of parameters in small NNs. However, the performance gained by using pre-training can't be achieved by using only standard L1/L2 regularization techniques.

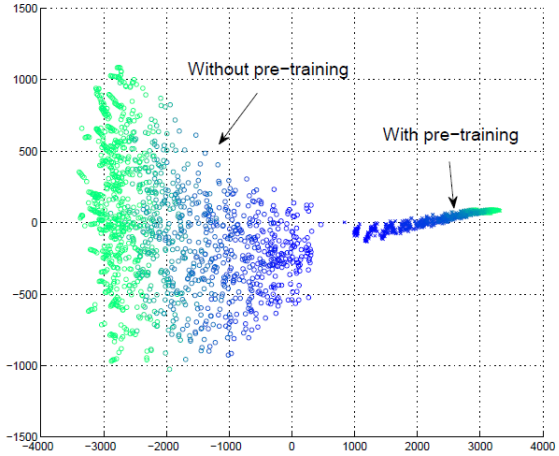
- Using 2D visualization techniques of network weights like tSNE and ISOMAP shows that the pre-trained network learns different weights in points of space disjoint from those learned without using pre-training as shown in figure 2.4. ISOMAP - which is a dimensionality reduction algorithm that preserves the global structure - shows that pre-trained NNs seem to be more similar and the self-similarity increases during training.

- They propose that non-convex optimization of NN weights is a complex process and the initial point largely affects the overall optimization performance. Thus, the use of pre-training helps in bringing the network weights to a starting point that leads to a better generalization performance.

- They propose the hypothesis that the unsupervised pre-training learns more robust disentangled representations of the factors of variations in the data. This makes it easier to combine these factors for classification. But the problem of unsupervised pre-training is that it has no hint of which factors are more important for classification. Thus, supervised training helps in extracting these factors after they are disentangled using unsupervised pre-training. There is a risk, however, that some factors are lost in the lower layers and are not preserved in higher layers. As a result, supervised training may not be able to recover them. This may explain the loss in performance when using pre-training with very deep networks or when a small number of units are used per layer.



(a) tSNE Visualization



(b) ISOMAP Visualization

Figure 2.4: 2D Visualization of network outputs trained with and without unsupervised pre-training. The networks outputs generated given the test set inputs are concatenated then dimensions are reduced to 2 using tSNE in (a) and ISOMAP in (b). tSNE preserves local structure and ISOMAP preserves global structure. The points are generated for each training epoch to show the trajectory through training with colors representing training epochs from dark blue to green. The figures are taken from [40].

Chapter 3

Automatic Speech Recognition

3.1 Introduction

Automatic speech recognition (ASR) aims at transcribing human speech into words. It is a very challenging task since human speech signals are highly variable due to various speaker attributes, different speaking styles, uncertain environmental noises and so on. Moreover, ASR needs to map variable length speech signals into variable length sequences of words or labels. The ASR problem can be formulated as follows:

$$L^* = \underset{L}{\operatorname{argmax}}\, p(L|\mathbf{X}) \tag{3.1}$$

where L is a hypothesized label (word or phone) sequence and X is the input speech observation sequence that represents the given speech utterance where $\mathbf{X} = [\mathbf{x}_1 ... \mathbf{x}_T]$. Hence, the task of an ASR system is to find the most probable label sequence L^* that represents the linguistic content of the given observation sequence. Usually, the speech recognition problem is

reformulated using Bayes' rule as follows:

$$L^* = \underset{L}{\operatorname{argmax}}\, p(L|\mathbf{X}) = \underset{L}{\operatorname{argmax}}\, \frac{p(\mathbf{X}|L)p(L)}{p(\mathbf{X})} = \underset{L}{\operatorname{argmax}}\, p(\mathbf{X}|L)p(L) \quad (3.2)$$

An ASR system has a number of components. First of all, the speech signal is recorded through a microphone and is represented in digital format. The speech signal is non-stationary and highly variable. Hence, the signal is divided into small chunks called frames so that the signal within each frame is more stationary (because the signal in one frame is usually a part of an acoustic unit) and its features can be represented using a fixed length feature vector. This process is called *feature extraction* and the component that performs feature extraction is called the *front end*. The output of the front end is the observation sequence $\mathbf{X}$, where each observation is a feature vector representing one frame.

Another important component of an ASR system is the acoustic model. The acoustic model aims at modeling the acoustic properties of different speech units and modeling the relation of the acoustic features of the whole speech utterance with the utterance label sequence or text. This relation is formulated as $p(\mathbf{X}|L)$ in Eq. 3.2. A popular and successful model for the speech acoustic signal is the *hidden Markov model* (HMM). The HMM is very flexible in modeling variable length sequences and tolerates differences in speaking speeds. It models the likelihood of generating the speech observation sequence given a sequence of latent (hidden) discrete states. A *Gaussian mixture model* (GMM) has been widely used to model the relation between an HMM state and the speech observations that belong to this state. More recently, better performance has been obtained when a *deep neural network* replaces the GMM in modeling state observations within the hybrid DNN-HMM model [41, 42].

The language model is another component that helps in improving the ASR performance. The language model specifies the allowed or more prob-

able word or label sequences. This is important to reduce errors when there is acoustic ambiguity or noises. The language model is represented by $p(L)$ in Eq. 3.2. N-Gram language models [43] are widely used in ASR systems. More recently, the recurrent neural network (RNN) language model is proved to further improve the performance of speech recognition [44]. The RNN has a form of memory by having recurrent connections with the previous states of the hidden units.

Based on the acoustic and language models, speech recognition is achieved by using a decoder. The decoder component searches for the best label sequence that maximizes the scores computed by the acoustic and language models for a given speech utterance. It corresponds to the argmax operation in Eq. 3.2. Since the number of possible label sequences grows exponentially with the length of the input feature sequence, the efficiency and the speed of the decoder are important characteristics of a successful ASR system.

3.2 Feature Extraction

The first step in speech recognition is the extraction of a sequence of feature vectors $\mathbf{X}$ that represents the input speech signal. There are different kinds of feature vectors. A good feature vector needs to contain all relevant information for classification of the input signal in a form that is suitable for the acoustic model. Moreover, it is desirable to discard all non-relevant information. This section will focus on the details of two examples of feature vectors that are widely used in academic and commercial ASR systems, namely *Mel frequency spectral coefficients* (MFSC) and *Mel-frequency cepstral coefficients* (MFCC).

The first step is to take small overlapping windows of the speech signal so that the signal is quasi-stationary and contains features that are localized to single speech units. Usually the windows are taken every 10 milliseconds (ms) and each window has a length of 25 ms. This speech window is multi-

plied by a Hamming window (or similar bell shaped functions) to reduce the effect of discontinuity at the two edges of the window. Since different speech units and sounds differ mainly in the distribution of energy along different frequencies, then it is a common practice to convert the speech time domain signal into a frequency domain signal using FFT analysis. Afterwards, filter bank analysis is performed to measure energy in a small number of frequency ranges. These filters are distributed along the Mel scale frequency to simulate the human ear sensitivity to frequency differences. In Mel frequency filter bank analysis, triangular filters as shown in Fig.3.1 are used. The energy in each filter is computed to estimate the MFSC feature. In practice, the log is taken to compute the log-MFSC features which are more similar to the sensitivity of the human ear and allows a better discrimination ability. The log-MFSC feature works well in practice with neural network based models especially convolutional neural networks. However, the log-MFSC features are highly correlated because the spectral envelop changes smoothly along frequency and neighboring filters have some overlap. Hence, the discrete cosine transform is taken to reduce this correlation to generate MFCC features. This is important to achieve good performance with Gaussian Mixture Models (GMMs) especially when a diagonal covariance matrix is used.

In practice, the higher MFCC coefficients are discarded since most of information related to the vocal tract shape exists in the lower coefficients, while other information related to the voice pitch and intonation are concentrated in the higher coefficients. While this reduces variability, it also loses some information, which may be one reason why lower performance is observed when MFCC features are used instead of MFSC features with models such as DNNs that do not require uncorrelated features.

A simple feature that is added to the MFCC or the MFSC is the frame log energy. Moreover, it can be normalized to have a maximum value of one which leads to slight performance improvements in practice when the speech signal is clean. Otherwise, the energy can be normalized to have a zero mean.

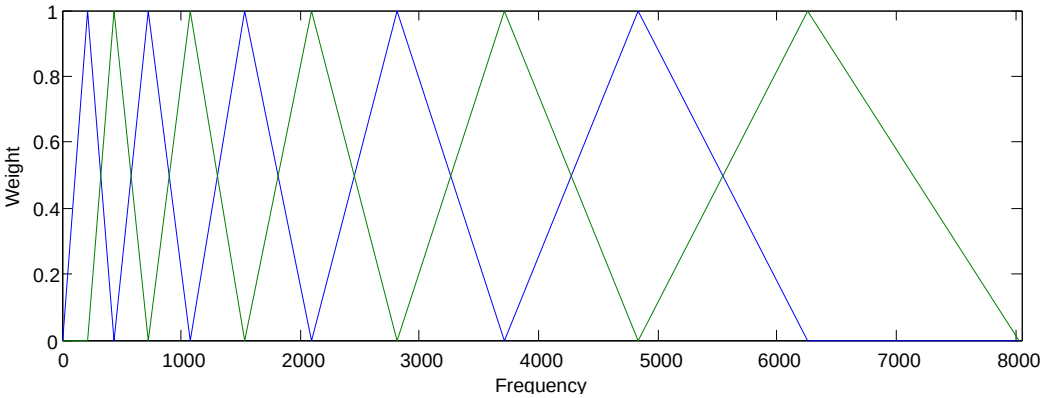


Figure 3.1: Mel scale filter bank analysis. The figure shows the boundaries of 10 filters. Each triangle represents the weights of one filter in the frequency domain based on Mel scale.

An HMM processes a sequence of frames, where each feature vector is computed from the speech signal within the frame boundaries. In practice, it is found that adding information about the temporal dynamics of the features is beneficial. This is done by adding the first and second order *deltas*. The first order delta Δx_t is computed as:

$$\Delta x_t = \frac{\sum_{\theta=1}^{\Theta} \theta(x_{t+\theta} - x_{t-\theta})}{2\sum_{\theta=1}^{\Theta} \theta^2}, \tag{3.3}$$

where Θ controls the size of the analysis window of delta and it usually takes the value of one or two. In this work, Θ is always set with two. The second order delta $\Delta^2 x_t$ is computed in the same way but using the values of Δx_t.

Throughout this dissertation, MFCC features are used with GMM based models, while log-MFSC features are used with neural network based models. In all cases, the first and second order deltas are appended in addition to the frame log energy. For MFCC features, 13 coefficients are computed from 20 Mel frequency filter bank filters (by taking the first 13 coefficients after applying the DCT). For log-MFSC features, 40 coefficients/filters are used.

3.3 The Hidden Markov Model

The *Hidden Markov Model* (HMM) is a statistical model suitable for modeling variable length sequential data especially time varying signals. It has been used in ASR systems since the seventies [45] and it is currently the most widely used model for ASR. Moreover, it has been used successfully for other applications that use sequential data such as DNA [46], protein analysis [47], and handwriting recognition [48].

The HMM is a finite state machine. At each discrete time instant t, the HMM is assumed to be in one state i and generates an observation o_t depending on the output probability distribution of this state $b_i(o_t)$. The HMM states are assumed to be unobserved and hidden. At each time instant, the HMM transitions from one state j to another state i (it can be the same state) with a probability a_{ij}. The HMM makes the following assumptions:

- First-order Markov assumption: The probability of transitioning to a certain state depends only on the previous state.

$$p(q_{t+1} = j | q_t = i, q_{t-1} = k, ...) = p(q_{t+1} = j | q_t = i) = a_{ij} \tag{3.4}$$

- Conditional independence of observations: This means that an observation at time t is independent from other observations given the state at time t. That is:

$$p(o_t | o_1 ... o_{t-1} o_{t+1} ... o_T, q_1 = k, ..., q_t = i, ..., q_T = j) = p(o_t | q_t = i) = b_i(o_t) \tag{3.5}$$

These assumptions help in efficient computation of the observation sequence likelihood and training of the HMM model. However, consecutive speech frames are highly correlated so these are not realistic assumptions about speech signals. Despite these discrepancies, the HMM works very well in practice in ASR applications.

The HMM can have a number of different topologies of state transitions. In the ergodic topology, all states are connected together. In ASR applications, a left to right topology such as the one shown in figure 3.2 is used, where each state represents an acoustic unit in a word or an utterance and the HMM is not allowed to move backwards. The HMM is allowed to transition either towards the state on the right or to stay in the same state. In this way, each state generates one or more observations or frames.

The observations of the HMM can be discrete or continuous. Speech features are normally continuous and multivariate. The *Guassian Mixture Model* (GMM) is a suitable model for the state output and it is widely used in ASR systems. The GMM defines the output probability distribution as:

$$b_i(\mathbf{o}) = \sum_k c_{ik} \mathcal{N}(\mathbf{o}, \mu_{ik}, \Sigma_{ik}), \tag{3.6}$$

where $\mathcal{N}(\mathbf{o}, \mu, \Sigma)$ is the Gaussian distribution and it is defined as:

$$\mathcal{N}(\mathbf{o}, \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^D |\Sigma|}} \exp\left(-\frac{1}{2}(\mathbf{o}-\mu)^\top \Sigma^{-1} (\mathbf{o}-\mu)\right), \tag{3.7}$$

and $\mathbf{o} \in \mathbf{R}^D$.

There are three problems to be solved when using HMMs:

1. Estimation of the likelihood of a given observation sequence $p(\mathbf{O}|\lambda)$
2. Estimation of the best state sequence $\mathbf{q}^*$: $\mathbf{q}^* = \text{argmax}_{\mathbf{q}}\, p(\mathbf{O}, \mathbf{q}|\lambda)$
3. Estimation of the model parameters that maximize the likelihood of the training data $\lambda^* = \text{argmax}_\lambda \prod_d p(O_d|\lambda)$

3.3.1 Estimation of the observation sequence likelihood

Given an observation sequence $\mathbf{O} = [\mathbf{o}_1, \mathbf{o}_2 ... \mathbf{o}_T]$, we would like to estimate the likelihood that this observation sequence was generated from a certain

model λ, that is $p(\mathbf{O}|\lambda)$. Since the model state sequence is hidden, all possible state sequences should be considered:

$$p(\mathbf{O}|\lambda) = \sum_{\mathbf{q}} p(\mathbf{O}, \mathbf{q}|\lambda) \tag{3.8}$$

$$= \sum_{\mathbf{q}} \prod_{t} p(q_t|q_{t-1}) p(\mathbf{o}_t|q_t) \tag{3.9}$$

$$\tag{3.10}$$

where $\mathbf{q}$ is the state sequence and $\mathbf{q} = [q_1, q_2, ..., q_T]$. There are an exponentially large number of possible state sequences. It would be impractical to directly sum over all of them. But, since it is a summation of multiplied terms and different state sequences share many terms, the multiplication distribution property can be used to reduce the number of computations. The forward algorithm does this. The forward probability $\alpha_j(t)$ is defined as the sum of the likelihoods of partial paths that end in state j at time t. It can be computed recursively as:

$$\alpha_j(t) = \sum_{i} \alpha_i(t-1) a_{ij} b_j(\mathbf{o}_t) \tag{3.11}$$

Assuming that the model is allowed to start only at one start state s and ends at one terminal state e, the initial values will be:

$$\alpha_s(1) = b_s(\mathbf{o}_1) \tag{3.12}$$

$$\alpha_i(1) = 0 \text{ for } i \neq s \tag{3.13}$$

Finally, the observation sequence likelihood is:

$$p(\mathbf{O}|\lambda) = \alpha_e(T) \tag{3.14}$$

Similarly, the likelihood can be computed recursively in the backward

direction by computing the backward probability $\beta_i(t)$, which is the likelihood of generating the observations $o_{t+1}...o_T$ and being at state i at time t. This backward probability plays an important role in learning of HMM parameters. The backward probability can be computed recursively as follows:

$$\beta_i(t) = \sum_j \beta_j(t+1) a_{ij} b_j(\mathbf{o}_{t+1}) \tag{3.15}$$

with the initial conditions

$$\beta_e(T) = 1 \tag{3.16}$$

$$\beta_i(T) = 0 \text{ for } i \neq e \tag{3.17}$$

and final likelihood computation as:

$$p(\mathbf{O}|\lambda) = \beta_s(1) b_s(\mathbf{o}_1) \tag{3.18}$$

3.3.2 Finding the best state sequence

The best state sequence is used in many applications, in training of the HMM model and in decoding of speech input to find the best state sequence that maps to the label sequence as will be described later.

The best state sequence can be optimally found using the Viterbi algorithm. The Viterbi algorithm recursively computes the following quantities:

$$\delta_j(t) = \max_i \delta_i(t-1) a_{ij} b_j(\mathbf{o}_t) \tag{3.19}$$

$$\psi_j(t) = \operatorname*{argmax}_i \delta_i(t-1) a_{ij} \tag{3.20}$$

where $\delta_j(t)$ is the likelihood of the best state sequence from time 1 to t and ends at state j, and $\psi_j(t)$ holds the previous state of this sequence at time

$t-1$. The algorithm is initialized as:

$$\delta_s(1) = b_s(\mathbf{o}_1) \tag{3.21}$$

$$\delta_i(1) = 0 \ \text{ for } \ i \neq s \tag{3.22}$$

At the end, the best state sequence $\mathbf{q}^*$ can be found by backtracking from state e and time T, until reaching the start state s at time 1 as follows:

$$q_T^* = e; \tag{3.23}$$

$$q_t^* = \psi_{q_{t+1}^*}(t+1) \ \text{ for } \ t = t-1, t-2, ..., 1 \tag{3.24}$$

and the likelihood of this best state sequence is:

$$p(\mathbf{O}, \mathbf{q}^*|\lambda) = \delta_e(T) \tag{3.25}$$

3.3.3 Learning model parameters

A common approach to learn the HMM model parameters is to use the *Maximum Likelihood Estimation* (MLE). Assuming that a model λ is to be trained on a training data set $\mathcal{D} = \{\mathbf{O}^{(1)}, ..., \mathbf{O}^{(N)}\}$, where $\mathbf{O}^{(n)}$ is the nth training sequence. The MLE training is achieved by finding the model parameters λ^* that maximize the likelihood of training data:

$$\lambda^* = \underset{\lambda}{\text{argmax}} \prod_n p(\mathbf{O}^{(n)}|\lambda) \tag{3.26}$$

$$= \underset{\lambda}{\text{argmax}} \sum_n \log p(\mathbf{O}^{(n)}|\lambda) \tag{3.27}$$

There is no known analytical method that can estimate model parameters in Eq. 3.27 directly. Instead, *Baum-Welch* [49] algorithm is used which works iteratively to find a local maximum of the mentioned criterion. The Baum-Welch algorithm is considered a special case of the *Expectation Maximization* (EM) algorithm. The details of the Baum-Welch algorithm can be

found in [50]. Given a version of the HMM model λ^l with output likelihood modeled using a GMM, an iteration of the Baum-Welch algorithm computes an improved set of model parameter λ^{l+1} as follows: First, in the expectation step, the forward-backward algorithm is performed to compute the forward and backward probabilities. The following quantities are estimated based on the current model parameters λ^l:

$$\begin{aligned}\gamma_i^{(n)}(t) &= p\left(q_t = i|\mathbf{O}^{(n)}, \lambda^l\right) \\ &= \frac{\alpha_i^{(n)}(t)\beta_i^{(n)}(t)}{p(\mathbf{O}^{(n)}|\lambda^l)}\end{aligned} \tag{3.28}$$

$$\begin{aligned}\eta_{ij}^{(n)}(t) &= p\left(q_t = i, q_{t+1} = j|\mathbf{O}^{(n)}, \lambda^l\right) \\ &= \frac{\alpha_i^{(n)}(t)a_{ij}b_j(\mathbf{o}_{t+1}^{(n)})\beta_j^{(n)}(t+1)}{p(\mathbf{O}^{(n)}|\lambda^l)}\end{aligned} \tag{3.29}$$

In the previous equations, the superscript (n) represents the quantities related to the nth training sequence. Moreover, for a GMM model that has multiple Gaussians, the occupation probability can be computed for the mth Gaussian component in a state using:

$$\gamma_{im}^{(n)}(t) = \frac{\alpha_i^{(n)}(t)\beta_i^{(n)}(t)}{p(\mathbf{O}^{(n)}|\lambda^l)} \frac{c_{im}\mathcal{N}(\mathbf{o}_t^{(n)}, \mu_{im}, \Sigma_{im})}{\sum_k c_{ik}\mathcal{N}(\mathbf{o}_t^{(n)}, \mu_{ik}, \Sigma_{ik})} \tag{3.30}$$

In the maximization step, an improved set of parameters (the set of parameters number $l+1$) are estimated as follows:

$$a_{ij} = \frac{\sum_n \sum_t \eta_{ij}^{(n)}(t)}{\sum_n \sum_t \sum_k \eta_{ik}^{(n)}(t)} \tag{3.31}$$

$$\mu_{im} = \frac{\sum_n \sum_t \gamma_{im}^{(n)}(t)\mathbf{o}_t^{(n)}}{\sum_n \sum_t \sum_k \gamma_{ik}^{(n)}(t)} \tag{3.32}$$

$$\Sigma_{im} = \frac{\sum_n \sum_t \gamma_{im}^{(n)}(t) \left(\mathbf{o}_t^{(n)} - \mu_{im}\right) \left(\mathbf{o}_t^{(n)} - \mu_{im}\right)^\top}{\sum_n \sum_t \sum_k \gamma_{ik}^{(n)}(t)} \tag{3.33}$$

$$c_{im} = \frac{\sum_n \sum_t \gamma_{im}^{(n)}(t)}{\sum_n \sum_t \sum_k \gamma_{ik}^{(n)}(t)} \tag{3.34}$$

3.4 Language Modeling

The language model helps in improving the overall ASR performance by favoring more probable word sequences. Given a word sequence $W = [w_1, w_2, ..., w_K]$, the language model estimates the joint probability:

$$p(W) = p(w_1, w_2, ..., w_K) = \prod_k p(w_k | w_{k-1}, w_{k-2}, ..., w_1) \tag{3.35}$$

The N-gram language model simplifies the estimation of the joint probability by assuming independance of words that came N words or more before the current word. That is to take into account only the previous $N-1$ words as follows:

$$p(W) \approx p_{\mathrm{N-gram}}(W) = \prod_k p(w_k | w_{k-1}, ..., w_{k-N+1}) \tag{3.36}$$

where N is the degree of the model. Bigram and Trigram language models are commonly used. Increasing the degree of the model gives lower perplexity and higher accuracy. Though, in practice as N increases, much larger training data is needed to get good estimates of N-gram conditional probabilities. Moreover, many N-grams will not be seen in the training corpus which might appear during speech recognition. One method to solve this problem is to use *discounting* and *smoothing*. It allocates a small probability for unseen word sequences. Another method is *backoff*, which backs off to lower order N-grams for word sequences that did not appear enough number of times during training.

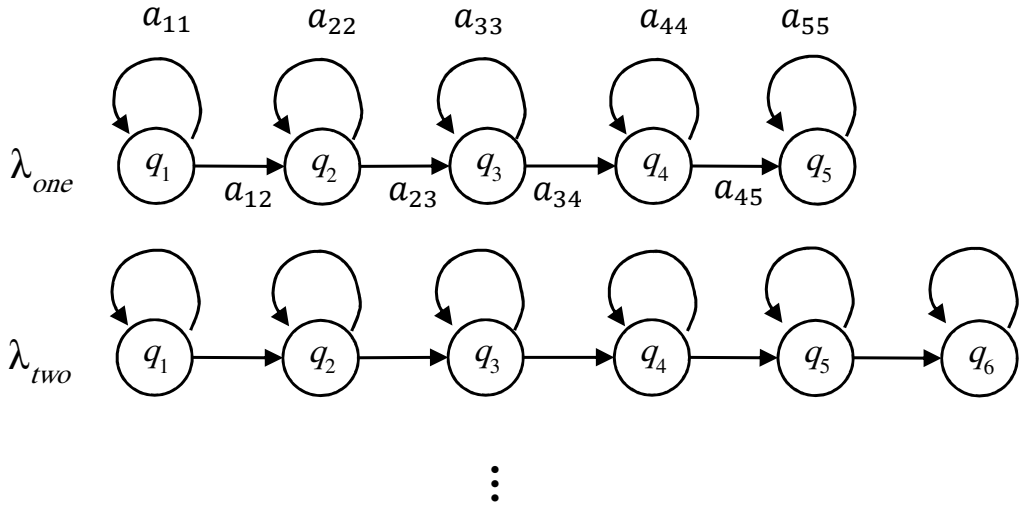


Figure 3.2: HMM word models for disconnected speech recognition.

3.5 HMM based ASR System

An HMM can be used for speech recognition in a number of ways. The simplest form is when an *isolated speech recognition* is used, where the task of the system is to classify separate words. In this case, each word is modeled using a different HMM model as shown in Fig. 3.2. Each HMM model has a number of states that's equivalent to the number of different acoustic parts of the word. To classify a given speech segment, the likelihood is computed given each model. The segment is classified by the word associated with the model that has the highest likelihood. In this case, the likelihood is either exactly estimated using the forward probability, or approximated with the likelihood of the best state sequence computed using the Viterbi algorithm.

A more advanced system allows connected (continuous) speech recognition, where the speaker can pronounce multiple words continuously. In this case, different word models are connected to form one big composite HMM as shown in Fig.3.3. A grammar defines the allowable transitions between different words. Moreover, a language model is usually used to give differ-

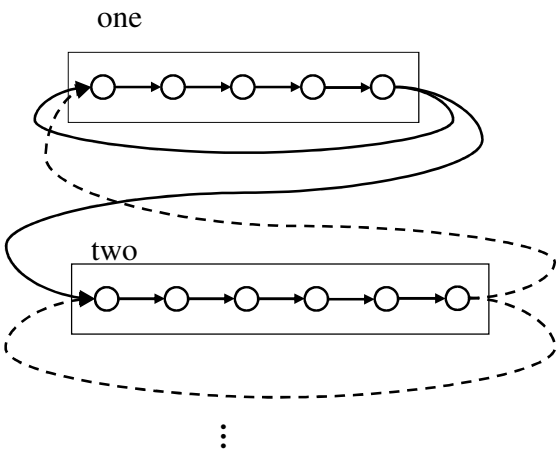


Figure 3.3: Composite HMM that is composed by connecting different word models.

ent weights to different word transitions. Performing speech recognition is achieved by finding the most likely state sequence and mapping this sequence into a word sequence representing the uttered sentence. The disadvantage of this model is that large amount of training data is needed to learn a good model of each word. Moreover, the model cannot recognize unseen words that did not appear in the training data. To solve theses problems, models that represent sub-word units like phonemes can be used. In these models, each phoneme is modeled as a sequence of three states which are meant to represent the transitional beginning and end of the phoneme in addition to the middle stationary state. Word models are formed by composing a sequence of phoneme models that represent the word as shown in Fig. 3.4. A dictionary is used to specify the phoneme models that represent a word. This model has the advantage of sharing training data of each phoneme among different words, hence reducing the amount of training data. Moreover, it can recognize unseen words if they have corresponding entries in the dictionary.

If each phoneme has one HMM model, the used models are called monophone models. In reality, a phoneme may be pronounced differently in differ-

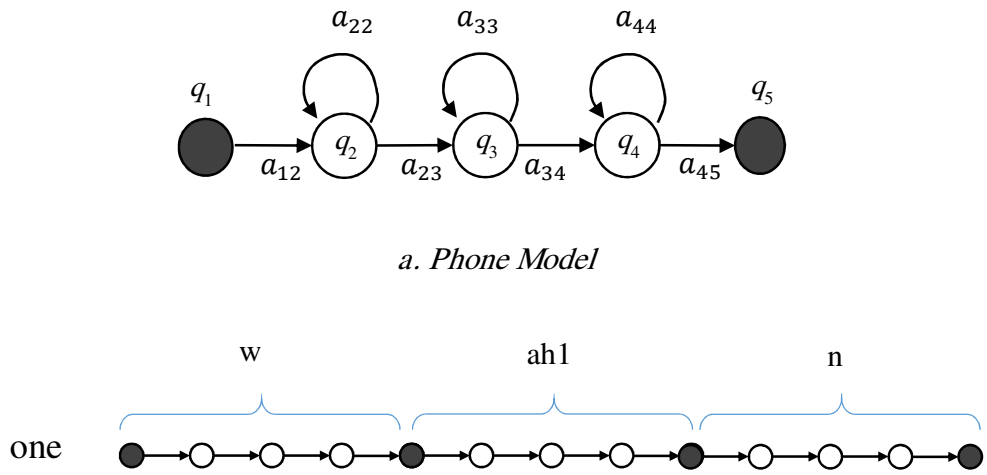


Figure 3.4: Word models composed by concatenating different phone models. a. shows a typical phone model that has three emmiting states and start and end non-emmitting states (colored black). b. shows a composite word model for "one" composed of three phonemes.

ent contexts. Triphone models solve this problem by using a different HMM model for each triphone context. Though, this results in a huge number of different models that are very difficult to obtain enough amounts of training data to learn each one of them. To solve this problem, states belonging to different triphones of the same phoneme that are similar acoustically are shared. Decision tree based clustering can be used to cluster states of different triphones based on the context. The number of generated state clusters can be controlled based on the available training data.

To train such a model, a composite left to right HMM model is created for each utterance by concatenating the states of all triphones that represent the label sequence of this utterance. The Baum-Welch algorithm is applied on these composite models on all training utterances to obtain a new improved

set of parameters. The training iterations are repeated a small number of times until convergence.

During speech recognition, all word models are composed from their triphones. Moreover, the words are connected with arcs that are weighted by the probability obtained from the language model. For N-gram language models, the decoder needs to keep track of word history during decoding. Hence, a huge lattice is built where different nodes represent different words with different histories as needed according to the N-gram language model. The decoder uses the Viterbi algorithm to find the state sequence that gives the maximum likelihood. This state sequence defines the desired output word sequence. Though, the decoding lattice may be so huge that special pruning algorithms are applied to speed up the decoding process.

3.6 Discriminative Training

In ASR, the aim is to minimize recognition error rates, like word error rate or phone error rate. However, MLE maximizes the training data likelihood. This may lead to sub-optimal performance. Moreover, the conditional independence assumption is known to be violated in speech signals. Additionally, the EM training of the HMM model finds a local maximum and does not guarantee optimality.

Alternatively, discriminative training can be used to directly optimize criteria more related to the ASR objective of minimizing the error rate or maximizing the accuracy of the recognition output. A number of different discriminative training criteria were proposed in literature.

A commonly used criterion is the *Maximum Mutual Information* (MMI) [51]. It maximizes the mutual information between the model prediction and the reference labels. This is actually achieved by maximizing the posterior probability of the reference label sequence. The MMI objective function can

be formulated as follows:

$$\mathcal{F}_{MMI}(\lambda) = \sum_n \log p\left(W_{ref}^{(n)}|\mathbf{O}^{(n)}, \lambda\right) \tag{3.37}$$

$$= \sum_n \log \frac{p\left(\mathbf{O}^{(n)}|W_{ref}^{(n)}, \lambda\right) p\left(W_{ref}^{(n)}|\lambda\right)}{\sum_W p\left(\mathbf{O}^{(n)}|W, \lambda\right) p\left(W|\lambda\right)} \tag{3.38}$$

The numerator represents the reference correct label sequence and the denumerator represents the summation of all possible label sequences. Hence, maximizing the MMI objective function requires maximizing the likelihood of the correct label sequence while minimizing all other competing sequences.

A number of other discriminative criteria were proposed that use slightly different objective functions. *Minimum Classification Error* was proposed in [52, 53], where a smooth error function on the sentence level is used and optimized. The work in [54] computed the error on the level of phonemes and words and used *Minimum Phone Error* (MPE) or *Minimum Word Error* (MWE) criteria. A more recent work employed *Large Margin* criteria as in [55, 56].

Since it is impractical to minimize all possible label sequences, an N-best list or a lattice of the competing hypotheses are used instead. A number of different optimization methods can be used to optimize the model parameters. A simple method is to use Gradient Ascent algorithm as in the early discriminative training work in [51]. Another more successful method is the *extended Baum-Welch Algorithm* (EBW) [57]. In practice, HMM parameters are estimated first using the MLE. Afterwards, the parameters are optimized using a few iterations of discriminative training.

3.7 Neural Network based ASR

Recently, deep neural network models (DNNs) gained renewed interest because they showed better performance on various speech recognition tasks

[58].

Early works of using NNs for acoustic modeling attempted to use the NN as a standalone model for recognizing simple speech units. In [59], Lippmann and Gold proposed a recurrent neural architecture, called *Viterbi net*, to emulate the work of the Viterbi algorithm. Hence it can recognize speech directly. However, it didn't have a training algorithm. The RNN weights were computed from a trained HMM model. The work in [60], proposed a neural network architecture called the *Time Delay Neural Network* (TDNN). The TDNN combines longer input contexts by a number of consecutive input frames and hidden activations to upper layers. The model was used to classify a small number of phonemes. Other similar models can be found in [61, 62, 63, 64, 65, 66].

Successful use of NNs for ASR was possible when a hybrid NN-HMM model is used [67, 68, 69, 70, 71, 72, 73, 74]. In one form of the model that became very successful recently, the NN replaces the GMM in scoring speech frames, and the HMM is used to model the sequential temporal behavior of the speech signal. The NN within this model predicts the posterior probability $p(q|\mathbf{x}_t)$ of an HMM state q given the frame $\mathbf{x}_t$. The frame likelihood is estimated using the Bayes' rule:

$$p(\mathbf{x}_t|q) = \frac{p(q|\mathbf{x}_t)p(\mathbf{x}_t)}{p(q)} \tag{3.39}$$

To train the hybrid model, state labels of each frame are needed. Usually in speech data sets, only phone or word sequences are available for each utterance, as it is hard to obtain frame labels for a large or a medium size training data set. Instead, a trained initial HMM model can be used to obtain label alignments using the Viterbi algorithm. The resulting frame labels are used afterwards to train the NN based on either the mean squared error or the cross entropy criteria.

In early works [68], the NN predicts the states of context independent

phones. Moreover, a window over input frames $\mathbf{y}_t = \mathbf{x}_{t-l}^{t+l}$ is used and leads to better frame classification and overall recognition performance. In this case, the system estimates frame likelihoods from the posterior probability $p(q|\mathbf{x}_{t-l}^{t+l})$. Despite the improved performance, this deviates more from the frame conditional independence assumption of the HMM. In other works, an RNN is used instead of the feed forward one (sometimes called a *Multilayer perceptron*) as in [75]. In this case, the RNN receives frames sequentially to classify each frame while taking into consideration the previous context. Moreover, the RNN can work bidirectionally by taking both previous and future frames. In more recent works [67, 76, 74], a context class based on the phonetic context is used to increase the power of the acoustic model. Context classes are found by clustering phonetic contexts found during training. A survey of similar models can be found in [77].

Another more recent approach that uses NNs is the *TANDEM* approach. This approach keeps using a GMM for modeling frame likelihoods within the HMM model. Instead of directly using MFCC features, it uses the NN to compute better features that are modeled using the GMM. In [78], the NN is trained to predict phoneme label distributions which may be augmented to other standard features like MFCC or PLP. Before feeding the NN outputs, a decorrelating transformation is applied to get features that are more suitable to a GMM with a diagonal covariance.

In [79], the so-called *bottleneck* features computed by a hidden layer of the NN is used instead of the output layer activations. The bottleneck features are computed by a hidden layer that has a relatively small number of nodes, so that the resulting feature vector is compact. The TANDEM approach has been applied to larger data-sets in [80, 81, 82]. Although the TANDEM approach does not benefit directly from posteriors computed by the NN other than as a feature vector as done in the hybrid models, it has the advantage of directly benefiting from all algorithms and models developed for the GMM model, like those used in speaker adaptation and GMM based discriminative

training.

Although all works mentioned above, the GMM-HMM models remained more popular and were used in the state of the art ASR systems. Only recently, NN based acoustic models received significant increase in interest due to recent improved results on a number of different ASR tasks [83, 84, 85, 86, 87, 88]. The work was motivated by the recent success of pre-trained deep neural network models used in different problems [89]. Initially, Mohamed et. al. [90] showed that a generatively pre-trained deep neural network hybrid HMM model can achieve a better performance than all previous results achieved by other models on the TIMIT phone recognition task. In this work, a *Deep Belief Network* (DBN) is trained to model the distribution of MFCC feature vectors of the speech training data set in an unsupervised way. The weights of the DBN are used as initial values of a similar feed forward NN. The DBN weights are learned by stacking the weights of multiple Ristricted Boltzmann Machines (RBMs). Each RBM is trained to model the features computed by the lower layer or the raw speech input. Thus, the process is called RBM based pre-training. The NN is used to predict posterior probabilities of HMM state labels of a mono-phone based model, where each phone is modeled by three states. The experiments show that more gains are obtained with deeper models that have in the range of five to eight hidden layers. Shortly thereafter, the work has been extended to several large vocabulary ASR tasks with triphone HMM models [85, 86, 91, 42, 92, 93, 58]. In these models, the DNN predicts the posterior probabilities of the states of tri-phone HMM models which are called *senons*. These models achieve better performance than the GMM based models for large vocabulary tasks. This resulted in wide adoption of these DNN based models in various ASR systems. Moreover, more improvement is obtained with sequence based discriminative training using criteria like maximum mutual information, minimum bayes risk, and minimum phone error [94, 95, 96, 97, 98]. Moreover, similar improvements has been obtained with

TANDEM bottleneck features approach when a DNN is used [99].

The improvement of hardware especially the use of Graphical Processing Units (GPUs) resulted in speedup of training of bigger and deeper DNN models in a way that would take prohibitively long time in the past. This contributes to the ability to better optimize the structure and meta-parameters of these DNNs. Moreover, these more powerful DNNs allow both using richer features like MFSC and classifying the input into one of a large number of classes that matches the number of senons in an HMM model. All these factors in addition to the use of pre-training and better training algorithms resulted in the described significant performance improvement in speech recognition.

Chapter 4

Convolutional Neural Networks for ASR

Speech signals can have many variations that are unrelated to the linguistic content in speech. For example, different speakers can have different voice characteristics. Even for the same speaker, many variations can happen. Examples of these variations include: speaking speed, pitch, formant frequencies due to differences in vocal tract length, and even background noise.

A number of different techniques have been proposed to handle these variations. These techniques can be divided into two classes: passive and active. Passive techniques try to normalize the speech variations. This can be done by designing special features, such as MFCCs, that discard information not related to the speech linguistic content. The HMM is tolerant to differences in speaking speed and hence normalizes variations in rate. Active techniques try to estimate some parameters that transform the speech signal or adjust the trained speech models to work better in the new conditions. For example, in some speaker adaptation techniques, the HMM model parameters are modified to match the characteristics of a new speaker. In other speaker adaptation techniques, the speech signal of a new speaker is transformed into a canonical speaker that has been used to train the acoustic

model. In both cases, the new model or transformation parameters are first estimated from some speech adaptation data obtained from the new speaker before performing speech recognition of this speaker.

The DNN, as shown in section 3.7, can learn to handle some variations and learn more speaker invariant features at the upper layers. These invariant features are learned implicitly in the DNN weights. However even better performance may be obtained if these variations are explicitly handled in certain ways. For instance, vocal tract length normalization (VTLN) explicitly warps the frequency axis based on a single learnable warping factor to normalize speaker variations in speech signals that result from different vocal tract lengths among different speakers. As shown in [84, 93], VTLN can further improve recognition performance of the DNN-HMM hybrid models when applied to process input features prior to DNN modeling.

Alternatively, these variations can be handled directly under the framework of neural networks by employing a special NN structure called the convolutional neural network (CNN) [100]. The main advantage of the CNN is that domain knowledge can be explicitly used to design the CNN structure to handle application-specific variations. CNNs have achieved the state of the art performance in a number of image recognition applications [101, 13, 102]. In these applications, CNNs are designed to exploit translation invariance and tolerate small shifts of image patterns along various directions. As opposed to regular NNs, CNNs use unique network structures such as weight sharing, local connectivity, and pooling. As a result, the CNN is more robust against object translation, small image deformations and noise as well as small rotations and scaling. These invariances are hard to learn automatically in standard NNs. More recently, CNN ideas have been similarly applied to speech processing. In [103] and [104], the CNN structure has been used to learn acoustic features in an unsupervised way. These works apply convolution over time to handle small time shifts for better speech feature representations for speaker, gender, and phonetic classification, but without

successful results on speech recognition tasks.

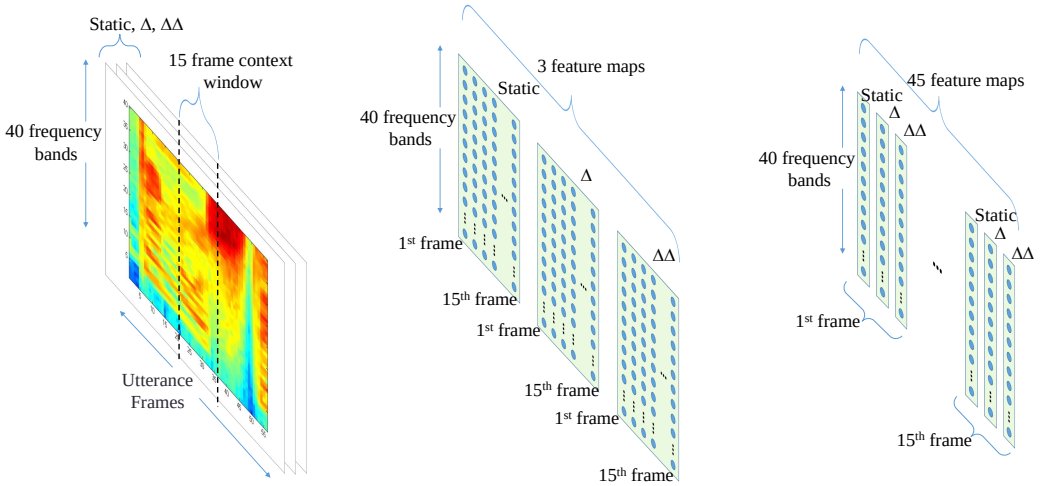
The CNN structure allows us to obtain features that are invariant to frequency differences when it is applied along the frequency axis [105, 106, 107]. Moreover, a CNN can handle temporal variations when it is applied along the time axis and hence better handle speaking speed differences [108]. However, the HMM can handle these variations within the hybrid NN-HMM model and reduces the benefit of applying convolution and pooling along time. This chapter describes the CNN structure and how it can be used for speech recognition. Moreover, a new CNN structure with a limited weight sharing (LWS) is described. This LWS better suits the speech signal where different patterns appear in different frequencies [109]. Then, this chapter presents experimental work for comparing different CNN and DNN structures. Finally, the conclusion is presented at the end.

4.1 Convolutional Neural Networks (CNN) for ASR

The convolutional neural network (CNN) is regarded as a successful variant of the standard neural network. Instead of using fully connected hidden layers, the CNN introduces a special network structure, which includes *convolution* and *pooling* layers to achieve translation invariance and tolerance to small deformations in patterns. The CNN was initially motivated by image processing and it has yielded excellent results in a number of image recognition tasks [100, 13].

4.1.1 Input Data Organization in CNN

The CNN requires the input data to be organized in a certain way. Each different feature is presented in a different feature map. A feature map represents the values of the same feature along different locations. For example,



a. Input utterance

b. Input features organized in 2D feature maps.

c. Input features organized in 1D feature maps.

Figure 4.1: Two different ways to organize speech input features to be fed into CNN. The above example assumes that 40 MFB features with first and second derivatives and a context window of 15 frames are used as the input.

in image processing applications, it is intuitive to organize each feature map as a two-dimensional (2-D) array including all pixel values in x and y (horizontal and vertical) coordinates. For color images, RGB (red, green, blue) values can be viewed as three different 2-D feature maps.

To process speech signals, we need to use features that are organized along frequency or time (or both) so that the convolution operation can be correctly applied. In this sense, the well-known MFCC features are not appropriate for convolution over frequency because the decorrelating discrete cosine transform projects the data into a new basis that doesn't enjoy locality along the frequency axis. In other words, each MFC coefficient represents a feature extracted from the whole frequency spectrum. On the other hand, the log-energy computed from a set of Mel filter banks (denoted as log-MFSC features) can be used because each value represents the energy in a different frequency band. Hence, we get data locality. In this chapter we

will assume using log-MFSC features as the CNN input. Moreover, their first and second temporal derivatives can be appended. And, similar to the framework described in chapter 3, features from a number of consecutive frames representing a context window of 11-15 frames are included as an input to the CNN.

There are several different methods to organize these log-MFSC features as different maps for a CNN. Firstly, as shown in Fig. 4.1.b, they can be arranged as three 2-D feature maps, corresponding to the static, the first derivative and the second derivative features. Each feature map represents the same feature along the frequency (filter band index) and time (frames within each context window) axes. In this case, a two-dimensional convolution is performed (explained later) to normalize both frequency and temporal variations at the same time.

A different choice is to consider frequency variations only. In this case, a number of one-dimensional (1-D) feature maps are used. In this case, different frames and different feature orders (static, first, and second derivatives) belong to different feature maps. Each feature map represents the values of same feature along different frequency bands (filter bank indexes), as shown in Fig. 4.1.c. For example, if the context window contains 15 frames and 40 filter banks are used for each frame, we get 45 (15×3) 1-D feature maps and each map has 40 dimensions as shown in Fig. 4.1.c. In this case, 1-D convolution will be applied along frequency axis. In this chapter, we mainly focus on this arrangement and conduct 1-D convolution along frequency.

Once input feature maps are formed, convolution and pooling operations are applied to generate the convolution and pooling layers activations in sequence as in Fig. 4.2. Similar to the input layer, each one of them contains a number of feature maps as well. A pair of convolution and pooling layers in Fig. 4.2 is usually called one CNN layer. Obviously, more CNN layers can be added one by one to construct a deep CNN.

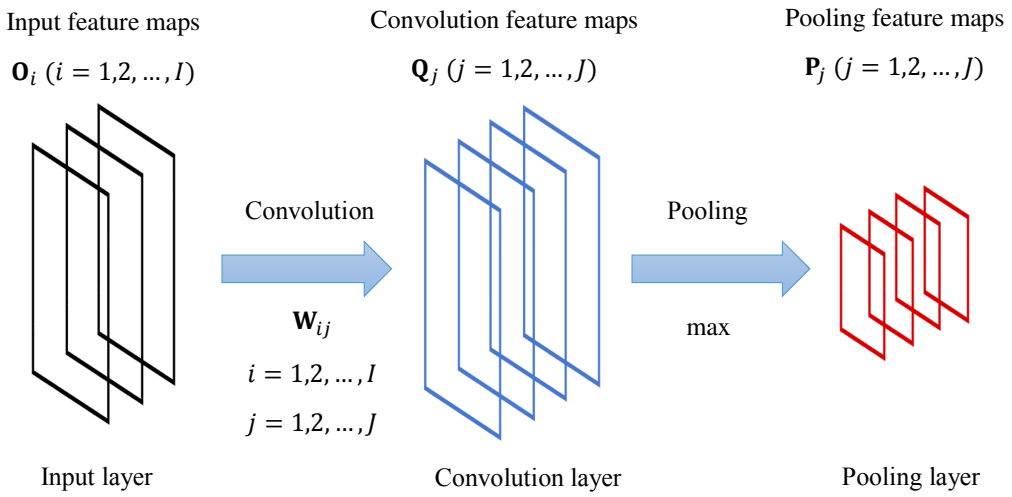


Figure 4.2: An illustration of one CNN layer consisting of a convolution layer and a pooling layer, where the mapping from the input layer to the convolution layer is based on Eq. 4.2 and the mapping from the convolution layer to the pooling layer is based on Eq. 4.3.

4.1.2 Convolution Layer

As shown in Fig. 4.2, all input feature maps (assume I in total), $O_i\,(i = 1, \cdots, I)$ are mapped into a number of feature maps (assume J in total), $Q_j\,(j = 1, \cdots, J)$ in the convolution layers based on a number of local filters ($I \times J$ in total), $\mathbf{w}_{ij}\,(i = 1, \cdots, I; j = 1, \cdots, J)$. The mapping can be represented as the well-known convolution operation in signal processing. Assuming input feature maps are all one dimensional, each unit of one feature map in the convolution layer can be computed as:

$$q_{j,m} = \sigma(\sum_{i} \sum_{n=1}^{F} o_{i,n+m-1} w_{i,j,n} + w_{0,j}), \tag{4.1}$$

where $o_{i,m}^{(c)}$ is the m-th unit of the i-th input feature map O_i, $q_{j,m}$ is the m-th unit of the j-th feature map Q_j of the convolution layer, $w_{i,j,n}$ is the nth element of the weight vector, $\mathbf{w}_{i,j}$, connecting the ith feature map of the input

to the jth feature map of the convolution layer, and F is called the filter size which is the number of input bands that each unit of the convolution layer receives. The previous equation can be written as a more concise matrix form using the convolution operator $*$ as:

$$Q_j = \sigma(\sum_i O_i * \mathbf{w}_{i,j} + w_{0,j}) \tag{4.2}$$

where O_i represents the i-th input feature map and $\mathbf{w}_{i,j}$ represents each local filter with the weights flipped to adhere to the convolution operation definition. Both O_i and $\mathbf{w}_{i,j}$ are vectors if one dimensional feature maps are used or they are matrices if two dimensional feature maps are used as described in the previous section (where 2-D convolution is applied to the above equation). The number of feature maps in the convolution layer depends on how many sets of local filters are used in the convolutional mapping. Obviously, feature maps become smaller after each convolution operation, i.e., each dimension decreases by the filter size minus one due to convolution. On the other hand, if we want to get the same size of the feature maps after convolution, the input feature maps can be padded with dummy locations on both sides (dummy frequency bands and/or frames).

A convolution layer is different from a standard fully connected layer in a number of aspects. Firstly, each unit receives input from a local area of the input, so the computed features have a locality property and thus each unit represents features of a local region of the input. Secondly, the units are organized in a number of feature maps, where all units in the same feature map share the same weights but receive input from different locations of the lower layer. Each feature map computes one feature of the input over all possible locations by applying the local filter defined by the map weights to the input through the convolution operation. This weight sharing scheme is called full weight sharing. Other weight sharing schemes will be described later in the chapter.

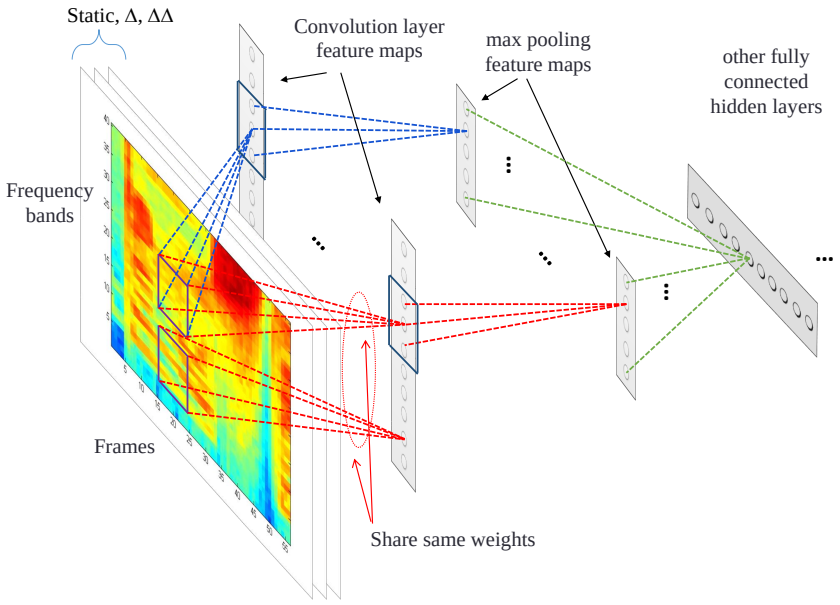


Figure 4.3: An illustration of the regular CNN that uses the full weight sharing scheme, where 1-D convolution is applied along frequency bands.

4.1.3 Pooling Layer

As shown in Fig. 4.2, a pooling operation is applied to the convolution layer to generate the pooling layer above each convolution layer. The pooling layer is organized as a number of feature maps that is equal to the number of the feature maps in the convolution layer. The pooling layer serves two purposes. Firstly, it reduces resolution of feature maps to minimize the number of values to be fed into upper layers. Secondly, it adds invariance to small variations in location. This is achieved by applying some pooling function at every location of the convolution feature map. The pooling function computes some overall property of a local region by using a simple function like *maximization* or *averaging*. The pooling function is applied to each convolution feature map independently so that each pooling unit applies the pooling function to a local range. If the max-pooling function is used, the pooling layer is defined

as:

$$p_{i,m} = \max_{n=1}^{G} \quad q_{i,(m-1)\times s+n} \tag{4.3}$$

where G is the pooling size, and s is a sub-sampling factor representing the shift between adjacent pooling regions. Similarly, if the average function is used, the output is calculated as:

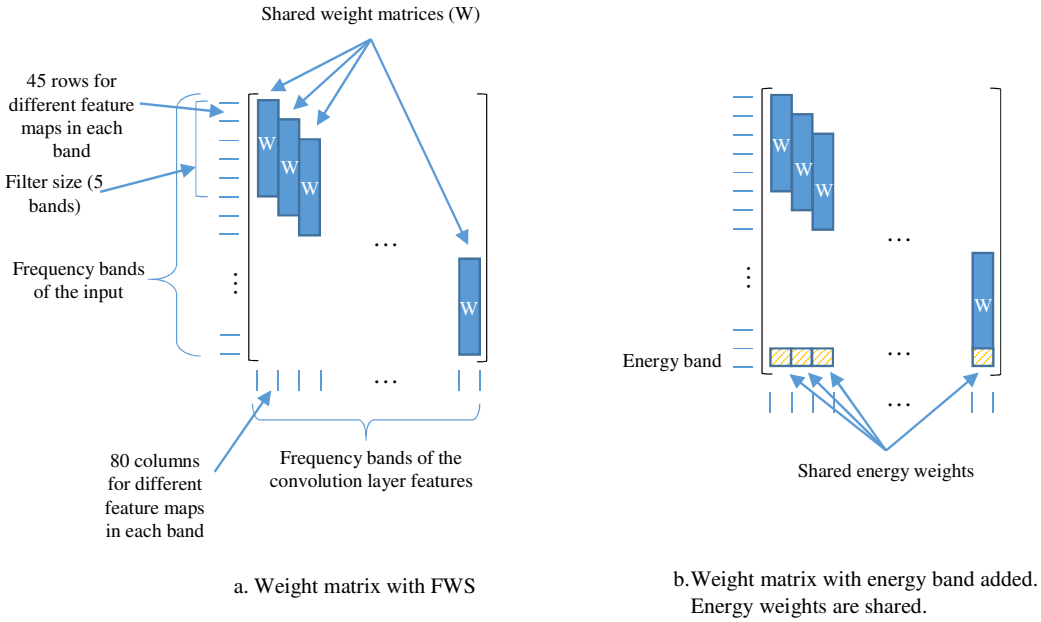
$$p_{i,m} = r \sum_{n=1}^{G} \quad q_{i,(m-1)\times s+n} \tag{4.4}$$

where r is a scaling factor that can be learned. It has been shown that max-pooling performs better than the average function in image recognition applications [110]. In image processing applications, G and s are usually set to have the same value, but in this work, they are adjusted independently. Moreover, a non-linear activation function can be applied to the above $p_{i,m}$ to generate the final output. Figure 4.3 shows a pooling layer with a pooling size of 3. Each pooling unit receives input from three convolution layer units in the same feature map. As a result, the pooling layer has the same number of feature maps but with a lower resolution and it becomes more invariant to small shifts in input features.

4.1.4 Learning of CNN layers

All weights in the convolution layer can be learned using the same error back-propagation algorithm but some special treatments are needed to take care of sparse connection and weight sharing. In order to illustrate the learning algorithm for CNN layers, let's first represent the convolution operation in Eq. 4.2 in the same mathematical form as the fully connected NN layer so that the same learning algorithm in section 2.2 can be similarly applied.

If one dimensional feature maps are used, the convolution operations in Eq. 4.2 can be represented as a simple matrix multiplication by introducing a large sparse weight matrix $\hat{\mathbf{W}}$ as shown in Fig. 4.4, which is formed by



a. Weight matrix with FWS

b. Weight matrix with energy band added. Energy weights are shared.

Figure 4.4: All convolution operations in each convolution layer can be equivalently represented as one large matrix multiplication using a sparse weight matrix, where local connectivity and weight sharing are represented in the above matrix form. The above figure assumes filter size of 5 and 45 input feature maps, and 80 feature maps in the convolution layer. The matrix $\mathbf{W}$ has 45×5 rows (for 45 input feature maps and filter size of 5) and 80 columns (for the 80 feature mas of the convolution layer).

replicating a basic weight matrix $\mathbf{W}$ as in Fig. 4.4 a. The basic matrix $\mathbf{W}$

is constructed from all local filter weights, $\mathbf{w}_{i,j}$ as follows:

$$\mathbf{W} = \begin{bmatrix} w_{1,1,1} & w_{1,2,1} & \cdots & w_{1,J,1} \\ \vdots & \vdots & \ddots & \vdots \\ w_{I,1,1} & w_{I,2,1} & \cdots & w_{I,J,1} \\ \vdots & \vdots & \ddots & \vdots \\ w_{I,1,2} & w_{I,2,2} & \cdots & w_{I,J,2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{I,1,F} & w_{I,2,F} & \cdots & w_{I,J,F} \end{bmatrix}_{I \cdot F \times J} \tag{4.5}$$

where $\mathbf{W}$ has $I \cdot F$ rows and J columns where F denotes the filter size (the number of bands) and each band contains I rows for I input feature maps, and J is the number of feature maps in the convolution layer.

Meanwhile, the input and the convolution feature maps are also vectorized as row vectors $\hat{\mathbf{o}}$ and $\hat{\mathbf{q}}$. One single row vector $\hat{\mathbf{o}}$ is created from all input feature maps O_i $(i = 1, \cdots, I)$ as follows:

$$\hat{\mathbf{o}} = [\, \mathbf{v}_1 \,|\, \mathbf{v}_2 \,| \quad ... \quad |\, \mathbf{v}_M \,], \tag{4.6}$$

where $\mathbf{v}_m$ is a row vector containing the values of the mth frequency band along all I feature maps, and M is the number of frequency bands in the input layer. Therefore, the convolution layer outputs computed in Eq. 4.2 can be equivalently expressed by the sparse weight matrix as follows:

$$\hat{\mathbf{q}} = \sigma\big(\hat{\mathbf{o}}\hat{\mathbf{W}}\big) \tag{4.7}$$

This equation has the same mathematical form as a regular fully connected hidden layer as in Eq. 2.3. Therefore, the convolution layer weights can be updated using the back-propagation algorithm as in Eq. 2.7. The update for $\hat{\mathbf{W}}$ is similarly calculated as:

$$\Delta\hat{\mathbf{W}} = \epsilon \cdot \hat{\mathbf{o}}'\mathbf{e}. \tag{4.8}$$

where $\mathbf{e}$ is the back-propagated error coming from the upper layer. Since the weights are shared, the updates over these shared weights should be summed up as:

$$\Delta w_{i,j,n} = \sum_{m} \Delta \hat{\mathbf{W}}_{i+(m+n-2)\times I, j+(m-1)\times J} \tag{4.9}$$

where I and J are the number of feature maps in the input and convolution layers.

Since the pooling layer has no weights, no learning is needed here. However, the error signals should be back-propagated to lower layers through the pooling function. In the case of max-pooling, the error signal is back-propagated only to the most active (largest) unit among each group of pooled units. The error signal reaching the lower convolution layer can be computed as:

$$e_{i,n}^{\text{low}} = \sum_{m} e_{i,m} \cdot \delta(u_{i,m} + (m-1) \times s - n), \tag{4.10}$$

where $\delta(x)$ is the delta function and it has the value of 1 if x is 0 and zero otherwise, and $u_{i,m}$ is the index of the unit with the maximum value among the pooled units and it is defined as:

$$u_{i,m} = \underset{n=1}{\overset{G}{\text{argmax}}} \quad q_{i,(m-1)\times s+n} \tag{4.11}$$

4.1.5 Pre-training CNN Layers

RBM-based pre-training improves DNN performance especially when the training set is small. Pre-training initializes DNN weights to a proper range that leads to better learning. For convolutional structures, a convolutional RBM (CRBM) has been proposed in [18]. Similar to an RBM, training of a CRBM aims at maximizing the likelihood function of training data using the approximate contrastive divergence (CD) algorithm. In a CRBM, the convolutional layer activations are stochastic. A CRBM defines a multinominal distribution over the pooled hidden nodes. Hence, at most one unit in

each pooled set of units can be active. This requires either having no overlap between pooled units (i.e., the pooling size should be the same as the sub-sampling factor) or having different convolutional units attached to each pooling unit as in limited weight sharing as described in Sec. 4.2. Refer to [18] for more details on CRBM-based pre-training.

4.1.6 Treatment of Energy Features

In ASR, the log energy is usually calculated per frame and appended to other spectral features. In a CNN, it is not suitable to treat energy the same way as other filter bank energies since it is the sum of energy in all frequency bands and it is not local to any frequency. In this work, the log energy features are appended as extra inputs to all convolution units as in Fig. 4.4.b. Other non-localized features can be similarly treated in this way. Experimental results in section 4.3 show that using the energy feature consistently improves the overall system performance.

4.1.7 Overall Architecture

The basic building block of the CNN is a pair of hidden layers: a convolutional layer and a pooling layer. The input contains a number of localized features organized as a number of feature maps. The size (resolution) of feature maps gets smaller at upper layers as more convolution and pooling operations are applied. Usually one or more fully connected hidden layers are added on top to combine the features at all locations before feeding to the output layer.

In the hybrid NN-HMM model framework, the topmost layer is a softmax layer that computes the posterior probabilities for all HMM states. These posteriors are used to estimate the scaled likelihoods of all HMM states per frame by dividing by the state prior probabilities. Finally, the likelihoods of all HMM states are sent to a Viterbi decoder for recognition.

4.1.8 Benefits of Using CNNs for ASR

The CNN has three properties: locality, weight sharing, and pooling. Each of them has a good potential to improve speech recognition performance. Locality in convolution layer units allows more robustness against non-white noise where some frequency bands are cleaner than the others. This is because features computed in cleaner parts are less contaminated by noise. Noise only affects speech features in noisy frequency bands in the lower layers while noise can be better dealt with in the upper layers, which combine different frequency bands. This is clearly better than simply handling the noise in the lower layers as in the standard NNs. Moreover, locality reduces the number of NN weights to be learned and hence decreases overfitting. Weight sharing may improve model robustness and reduce overfitting as each weight is learned from all locations in the input instead of just one location. Both locality and weight sharing are needed for pooling. In pooling, the same feature values computed at different locations are pooled together and represented by one value. This leads to minimal differences in the computed features when the input patterns are shifted, especially when max-pooling is used. This is very helpful in handling small frequency shifts that are common in speech signals. These frequency shifts may result from differences in vocal tract lengths of different speakers. Even for the same speaker, small frequency shifts may often occur. These shifts are difficult to handle within other models like the GMM or the DNN where many Gaussians/hidden units are needed to handle all possible combinations of shifts in different frequencies. Moreover, it is difficult to learn an operation like max-pooling in standard NNs.

The same applies to temporal differences as well. In the hybrid NN-HMM model, a number of frames within a context window are usually processed by the NNs. Temporal variability due to varying speaking rate may be difficult to handle in standard NNs. CNNs can inherently handle this variability when convolution is applied along the context window frames. However, since the CNN computes an output for each frame for decoding, pooling and sub-

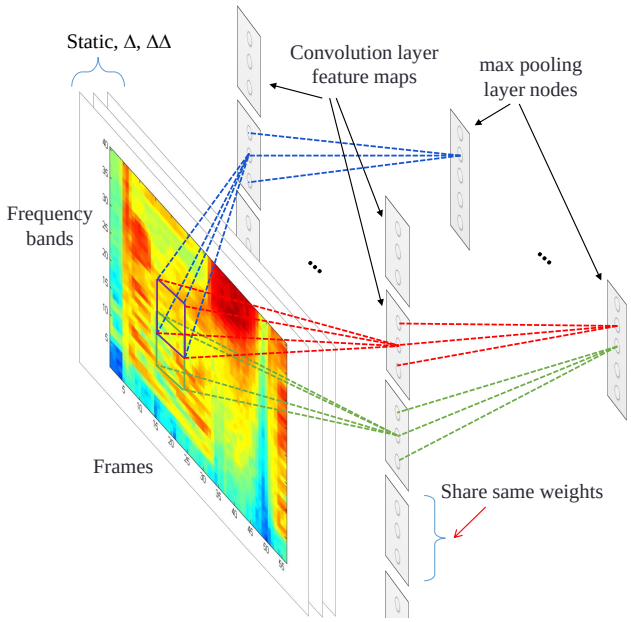


Figure 4.5: An illustration of CNN using the limited weight sharing scheme, where 1D convolution is applied along the frequency bands.

sampling may affect the fine temporal resolution seen by the higher layers of the CNN. A large pooling size may affect temporal localization of state labels. Hence, a suitable pooling size should be chosen to balance temporal invariance and temporal localization of output labels.

4.2 Limited Weight Sharing (LWS) CNN for ASR

4.2.1 Limited Weight Sharing (LWS)

We call the weight sharing scheme in Fig. 4.3, as described in the previous section, the *full weight sharing* (FWS) scheme. This is the standard CNN scheme used in image processing since the same patterns may appear at any location in an image. However, speech signals typically behave quite differ-

ently in various frequency bands. Using different sets of weights for different frequency bands may be more suitable since it allows the detection of different feature patterns at different frequencies. Fig. 4.5 shows an example of a *limited weight sharing* (LWS) scheme for CNNs, where only the convolution units that are attached to the same pooling unit share the same weights. These convolution units need to share the weights so that they compute the same features, but at different locations. The features are then pooled together in the upper pooling layer. In other words, each frequency band can be considered as a separate subnet with its own convolution weights. We call each of these subnets a section for notational convenience. Each section contains a number of feature maps in the convolution layer as well as in the pooling layer. But the pooling layer feature maps contain only one location while the convolution layer maps contain a number of locations that is equal to the pooling size. In mathematical terms, the convolution and pooling layer activations can be computed as:

$$q_{k,j,m} = \sigma(\sum_{i} \sum_{n=1}^{F} o_{i,(k-1)\times s+n+m-1} \cdot w_{k,i,j,n} + w_{k,0,j}) \tag{4.12}$$

where $w_{k,i,j,n}$ denotes n-th convolution weight, mapping from the i-th input feature map to the j-th convolution map in the k-th section, and m can take values from 1 up to G (pooling size). The pooling layer activations in this case can be computed using:

$$p_{k,j} = \max_{m=1}^{G} q_{k,j,m}. \tag{4.13}$$

Similar to the FWS convolution matrix representation, the above LWS convolution can also be represented as a matrix multiplication using a large sparse matrix as in Eq. 4.7 but both $\hat{\mathbf{o}}$ and $\hat{\mathbf{W}}$ need to be constructed in a slightly different way. First of all, the sparse matrix $\hat{\mathbf{W}}$ is constructed as Fig.

4.6, where each $\mathbf{W}_k$ is formed based on filter weights, $w_{k,i,j,n}$, as follows:

$$\mathbf{W}_k = \begin{bmatrix} w_{k,1,1,1} & w_{k,1,2,1} & \cdots & w_{k,1,J,1} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,I,1,1} & w_{k,I,2,1} & \cdots & w_{k,I,J,1} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,I,1,2} & w_{k,I,2,2} & \cdots & w_{k,I,J,2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,I,1,F} & w_{k,I,2,F} & \cdots & w_{k,I,J,F} \end{bmatrix}_{I \cdot F \times J} \qquad (k = 1, 2, \cdots, K) \qquad (4.14)$$

where these matrices $\mathbf{W}_k$ are different for each section and the same weight matrix is replicated G times within each section. Secondly, the convolution layer input is vectorized as described in Eq. 4.6 and the computed feature maps are organized as a large row vector $\hat{\mathbf{q}}$ by concatenating all values in each section as follows:

$$\hat{\mathbf{q}} = [\, \mathbf{v}_{1,1} \,|\, \ \ldots \ \ |\, \mathbf{v}_{1,G} \,|\, \ \ldots \ \ |\, \mathbf{v}_{K,1} \,|\, \ \ldots \ \ |\, \mathbf{v}_{K,G} \,], \qquad (4.15)$$

where K is the total number of sections, G is pooling size and $\mathbf{v}_{k,m}$ is a row vector containing the values of the units in the m-th band of the k-th section along all feature maps of the convolution layer as:

$$\hat{\mathbf{v}}_{k,m} = [\ \ q_{k,1,m}, \ \ q_{k,2,m}, \ \ \ldots \ \ q_{k,I,m} \ \ ] \qquad (4.16)$$

where I is the total number of input feature maps within each section.

Learning of the weights in the case of limited weight sharing can be done using the same eqs. (4.7) and (4.8) with $\hat{\mathbf{W}}$ and $\hat{\mathbf{q}}$ as defined above. Meanwhile, error vectors are propagated through the max pooling function as follows:

$$\mathbf{e}^{low}_{k,i,n} = \mathbf{e}_{k,i} \ \cdot \ \delta(u_{k,i} - n) \qquad (4.17)$$

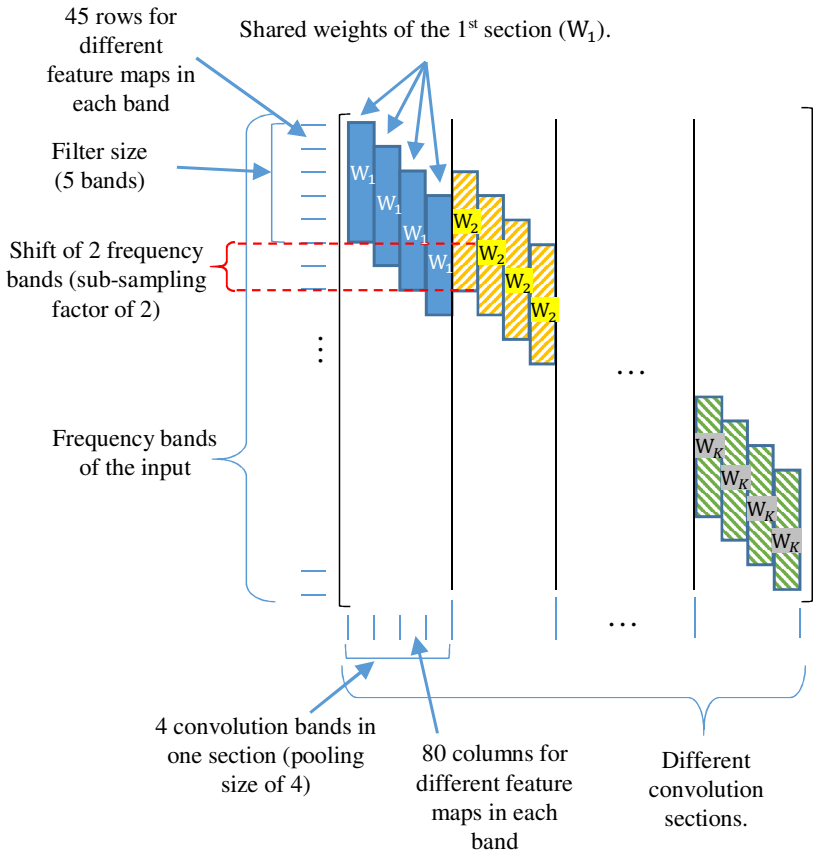


Figure 4.6: The CNN layer using limited weight sharing (LWS) scheme can also be represented as matrix multiplication using a large sparse weight where local connectivity and weight sharing are represented in the above matrix form. The above figure assumes filter size of 5, pooling size of 4, 45 input feature maps, and 80 feature maps in the convolution layer.

with

$$u_{k,i} = \underset{m=1}{\overset{G}{\operatorname{argmax}}} \quad q_{k,i,m}. \tag{4.18}$$

This LWS scheme also helps to reduce the total number of units in the pooling layer because each frequency band uses special weights that consider only the patterns appearing in the corresponding frequency range. As a result, a smaller number of feature maps per band should be sufficient. On

the other hand, this limited weight sharing scheme does not allow for addition of more convolution layers on top of it since the features in different pooling layer sections are unrelated and cannot be convolved locally. If it is necessary to have more than one convolution layer, however, it is possible to place a limited weight sharing convolution layer on top of a regular full weight sharing convolution layer.

4.2.2 Pre-training of LWS-CNN

In this section, we propose to modify the CRBM model in [18] for pre-training limited weight sharing (LWS) CNNs. For learning the CRBM parameters, we need to define the conditional probability of states of the hidden units given the visible ones and vice versa. The conditional probability of activation of a hidden unit $h_{k,j,m}$, that represents the m-th frequency band of the j-th feature map from the k-th section, given the CRBM input $\mathbf{v}$, is defined by the following softmax function:

$$P(h_{k,j,m} = 1|\mathbf{v}) = \frac{\exp\big(I(h_{k,j,m})\big)}{\sum_{n=1}^{p} \exp\big(I(h_{k,j,n})\big)}, \tag{4.19}$$

where $I(h_{k,j,m})$ is the summation of weighted signal reaching node $h_{k,j,m}$ from the input layer and it is defined as:

$$I(h_{k,j,m}) = \sum_{i} \sum_{n=1}^{f} v_{i,(k-1)\times s+n+m-1} w_{k,i,j,n} + w_{k,0,j} \tag{4.20}$$

The conditional probability distribution of $v_{i,n}$, which is the visible unit at the nth frequency band of the ith feature map, given the hidden unit states can be computed by the following Gaussian distribution:

$$P(v_{i,n}|\mathbf{h}) = \mathcal{N}(v_{i,n}; \sum_{j,(k,m)\in\mathbf{C}(i,n)} h_{k,j,m} w_{k,i,j,f(n,k,m)}\ , \sigma^2) \tag{4.21}$$

where the mean of the above distribution is the summation of the weighted signal arriving from the hidden units that are connected to the visible units, $\mathbf{C}(i,n)$ represents these connections as the set of indices of convolution bands and sections that receive input from the visible unit $v_{i,n}$, $w_{k,i,j,f(n,k,m)}$ is the weight on the link from the n-th band of the i-th input feature map to the m-th band and the j-th feature map of the k-th convolution section, $f(n,k,m)$ is a mapping function from the indices of connected nodes to the corresponding index of the filter element, and σ^2 is the variance of the Gaussian distribution, which is a fixed model parameter.

Based on the above two conditional probabilities, all connection weights of the above CRBM can be iteratively estimated by using the contrastive divergence (CD) algorithm. The weights of the trained CRBMs can be used as good initial values for the convolution layer in the LWS scheme. After the first convolution layer weights are learned, they are used to compute the convolution and pooling layer outputs using eqs. (4.12) and (4.13). The outputs of the pooling layer are used as inputs for pretraining the next layer, as is done in deep belief network training [41].

4.3 Experiments

In this section, experiments are conducted on two speech recognition tasks to evaluate the effectiveness of CNNs in ASR: TIMIT which is a small-scale phone recognition task, and VS which is a large vocabulary voice search task.

4.3.1 Experimental setup

The feature extraction is similar in these two data sets. Speech is analyzed using a 25-ms Hamming window with a fixed 10-ms frame rate. Speech feature vectors are generated by Fourier transform based filter bank analysis, which includes 40 log filter bank energy coefficients distributed on a Mel scale to generate the log-MFSC features, along with their first and second temporal

derivatives. Log energy per frame is added to the TIMIT feature vector, while VS does not use an energy component. All features are normalized over the whole training dataset so that each coefficient has zero mean and unit variance.

4.3.2 TIMIT Experiments

The TIMIT training dataset has 462-speaker, each of whom produces eight utterances (after discarding the two SA utterances from each speaker because they have the same labels and can bias the trained models). A separate set of 50 speakers is used for development, i.e. estimating the meta parameters that include language model scale, word insertion penalty, learning rate progress, and early stopping. Results are reported using the 24-speaker core test set, which has no overlap with the development set or training set.

Each of the 61 TIMIT phonemes is represented using a 3-state left-to-right HMM. The NN used in the hybrid NN-HMM model uses 183 classes which represent the HMM state labels. To prepare the NN targets, a mono-phone HMM model is trained on the training data set, and it is used to generate state-level labels based on forced alignment. For neural network training, learning rate annealing and early stopping strategies are utilized as in [41]. In the performed experiments a bi-gram language model is used to model the phonetic label sequence. For evaluation, the original 61 phoneme classes are mapped into a set of 39 classes as described in [111] for final scoring. This mapping is shown in table 4.1. HTK [112, 113] is used to compute speech features, and train the HMM and the language model. Special Matlab and C++ codes are written to train the NN models.

A number of experiments have been conducted on CNNs by using both the full weight sharing (FWS) and limited weight sharing (LWS) schemes. In this section, the performance of the CNN models is measured and compared against a DNN baseline. Moreover, the effect of changing different CNN parameters is studied as well. In these experiments, the following parameters

Table 4.1: The original 61 phonemes in the TIMIT dataset and their mapping into 39 classes

class	phonemes	class	phonemes
1	aa ao	21	l el
2	ae	22	m em
3	ah ax axh	23	n en nx
4	aw	24	ng eng
5	ay	25	ow
6	b	26	oy
7	ch	27	p
8	d	28	pau pcl tcl kcl q bcl dcl gcl epi h#
9	dh	29	r
10	dx	30	s
11	eh	31	sh zh
12	er axr	32	t
13	ey	33	th
14	f	34	uh
15	g	35	uw ux
16	hh hv	36	v
17	ih ix	37	w
18	iy	38	y
19	jh	39	z
20	k		

are used unless mentioned otherwise. The CNN has one pair of convolution and pooling layers and two fully connected hidden layers on the top. Each fully connected layer has 1000 units. The convolution and pooling parameters are: pooling size of 6, sub-sampling factor of 2, filter size of 8, and 150 feature maps for FWS and 84 feature maps per section for LWS. These parameters are selected because they led to good performance in preliminary experiments.

Effect of different CNN parameters

In this section, a set of experiments is conducted to measure the effect of changing various CNN parameters. The results are shown in figures 4.7–4.10. The figures show that both pooling size and the number of feature maps have the most significant impact on the final ASR performance. Fig. 4.7 shows that the configurations that has fixed pooling shift yield better performance by increasing the pooling size up to 6. LWS yields better performance with bigger pooling sizes. When there is no overlap between pooling windows (SS configurations), The improvement is not consistently better with bigger pooling sizes. Generally, overlapping the pooling windows result in slightly better performance; however, it leads to increased complexity of the model. Fig. 4.8 shows that a bigger number of feature maps usually leads to better performance especially with FWS. It also shows that LWS can achieve better performance with a smaller number of feature maps due to its ability to learn different filters for different frequency bands. This indicates that the LWS scheme is more efficient in terms of the number of hidden units. Fig. 4.9 shows that good performance is still obtained with smaller overlap which leads to simpler models which are smaller in size and computationally faster to use. It shows that as expected skipping bands by having a sub-sampling factor bigger than the pooling size harms the performance. Fig. 4.10 does not show consistent effect for different filter sizes. However, the performance starts to degrade with very big sizes starting from 16.

Energy Features

Table 4.2 shows the benefit of using energy features. It shows a significant improvement for adding energy features especially with FWS. Theoretically, energy can be estimated from the other MFSC features. Adding it to the convolution filters results in more discriminative power and provides a way to compare the local frequency bands processed by the filter with the overall spectrum.

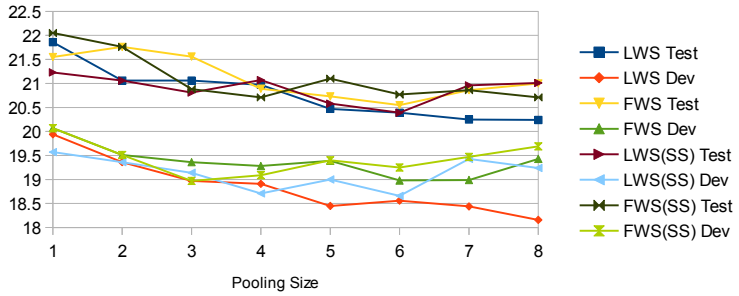


Figure 4.7: Effects of different pooling sizes of CNN on recognition performance (PER in %) for both limited weight sharing (LWS) and full weight sharing schemes (FWS). (SS) indicates that pooling windows do not overlap by using equal values for the pooling size and the sub-sampling factor.

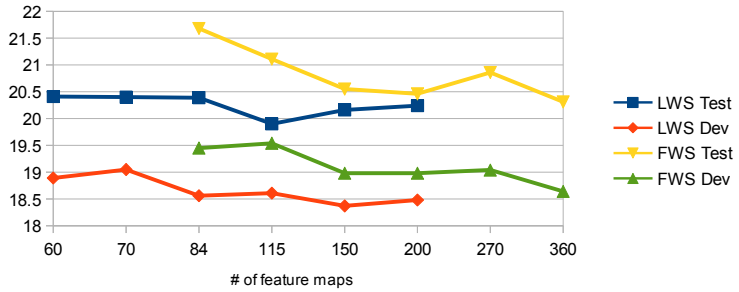


Figure 4.8: Effects of different numbers of feature maps of CNN on recognition performance (PER in %) for both limited weight sharing (LWS) and full weight sharing schemes (FWS).

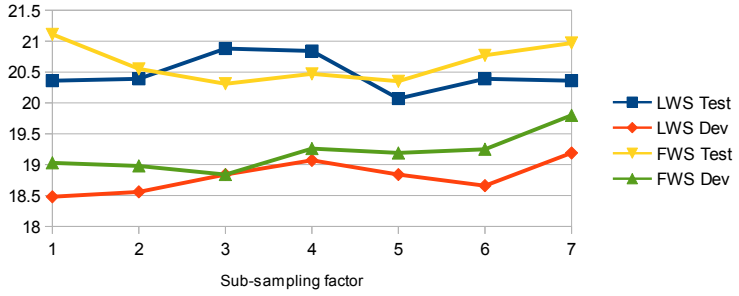


Figure 4.9: Effects of different sub-sampling factors of CNN on recognition performance (PER in %) for both limited weight sharing (LWS) and full weight sharing schemes (FWS).

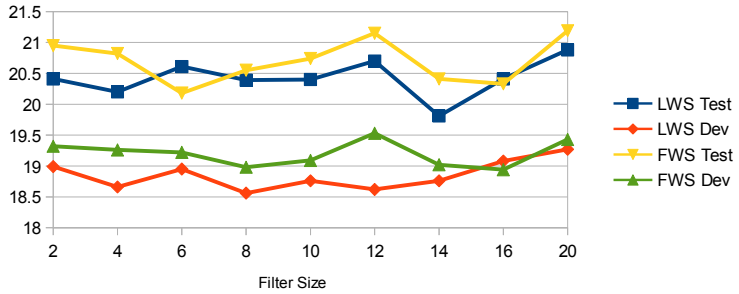


Figure 4.10: Effects of different filter sizes of CNN on recognition performance (PER in %) for both limited weight sharing (LWS) and full weight sharing schemes (FWS).

Table 4.2: Effect of using energy features on the core test set in PER.

	No Energy	Energy
LWS	20.61%	20.39%
FWS	21.19%	20.55%

Table 4.3: Performance of different pooling functions in PER.

	Average	Max
Development Set	19.63%	18.56%
Test Set	21.6%	20.39%

Pooling Function

Table 4.3 shows that max-pooling function performs better than the average function. The results are reported only for the LWS scheme. These results are consistent with what has been observed in image recognition applications [110].

Overall Performance

In this section, the overall performance of different CNN configurations and a baseline DNN is compared. All comparison results are listed in Table 4.4. To reduce variability due to different random initialization of the weights,

the table shows the performance in PER averaged over three models trained from different initial weights. The first row shows recognition performance of a DNN that has three hidden layers. Its first layer has 2000 units (to match the number of units in the CNNs). The other hidden layers have 1000 units each. The parameters of the CNNs in rows 2 and 3 are chosen based on the performance obtained in the previous sections on the development set. Both have a filter size of 8, a pooling size of 6, and a sub-sampling factor of 2. The number of feature maps is 150 for LWS and 360 for FWS. The table shows that CNN performance is much better than that of the DNN. LWS is slightly better than FWS even with less than half the number of units in the pooling layer. Although the number of parameters in the LWS convolution layer is slightly bigger than that of the FWS, the LWS-CNN gives a much smaller model size since LWS results in fewer weights in the upper fully connected layer. The CNN with LWS brings more than 8% relative reduction in PER versus the DNN. The last two rows show the performance of using two CNN layers. The fourth row shows the performance of using FWS for both convolution layers, and the fifth row shows that of using LWS in the second convolution layer. The parameters of the two CNN layers are only coarsely tuned on the development set. Although they do not show improvements over using one CNN layer, they use a much smaller number of parameters.

Statistical significance tests are performed to confirm the results shown in Table 4.4. The performed test is a form of matched pair test that is called matched pairs segment sentence word error rate [114]. The tests were conducted using the NIST SCTK scoring toolkit which is developed by the American National Institute of Standards and Technology (NIST). The test showed significance statistical differences between the CNN (both FWS and LWS) and the DNN models at a level of p=0.001. On the other hand, the test showed that the difference between the LWS-CNN and FWS-CNN is not statistically significant at a level of p=0.05. To conduct the tests, the best

Table 4.4: Performance on TIMIT of different CNN configurations, compared with DNNs, along with the size of the model in total number of parameters, and the speed in total number of multiply-and-accumulate operations. Average PERs were computed over 3 runs with different random seeds.

ID	Network structure	Average PER	# params	# ops
1	DNN	22.02%	6.9M	6.9M
2	CNN (LWS)	20.17%	5.4M	10.7M
3	CNN (FWS)	20.31%	8.5M	13.6M
4	CNN (FWS + FWS)	20.23%	4.5M	11.7M
5	CNN (FWS + LWS)	20.36%	4.1M	7.5M

performing CNNs and DNNs of the performed three runs are used which have the PERs of 21.86%, 20.16%, and 19.92% for the DNN, FWS-CNN, and LWS-CNN in order.

4.3.3 Large Vocabulary Results

This section shows the results of experiments conducted to measure the CNN performance on a large vocabulary ASR task. A voice search dataset (VS) containing 18 hours of speech data is used. Initially, a conventional tied-state triphone GMM/HMM is built. The HMM state labels are used as the targets for training the DNN and the CNN, where the training procedure of both the DNN and the CNN follows a standard recipe. First 15 iterations are run with a learning rate of 0.08 and then 10 more epochs are run with a learning rate of 0.002. In this section, the effect of pre-training is investigated as well. Pre-training is done using an RBM for fully connected layers and using a CRBM (as described in section 4.2.2) for convolution and pooling layers.

Table 4.5 shows that using a CNN improves the performance over a DNN whether pre-training is used or not. Similar to TIMIT, the CNN improves performance by about 8% relative over the DNN for the VS task without pre-training. With pre-training, the relative reduction in PER is about 6%. Moreover, the table shows that pretraining of the CNN can improve ASR

Table 4.5: Performance on the VS large vocabulary data set in WER% with and without pre-training (PT).

	No PT	With PT
DNN	37.1%	35.4%
CNN	34.2%	33.4%

performance although the effect of pre-training on the CNN is not as dramatic as it is on the DNN.

4.4 Conclusion

This chapter has presented a concise description of how to apply CNNs to speech recognition in such a way that CNN structures can explicitly handle certain speech variations. It has been shown that it is more beneficial to apply convolution through the frequency axis since it leads to models that are invariant to small frequency shifts, which normally occur in speech signals due to speaker differences. In addition, a new limited weight sharing scheme has been proposed which can handle speech features in a better way. Experimental results show that this limited weight sharing provides a slightly better performance than the standard full weight sharing scheme originally used in image recognition applications. Moreover, the limited weight sharing scheme leads to a much smaller number of units in the pooling layer, resulting in a smaller model size and lower computational complexity versus the full weight sharing scheme. The experimental results show that CNNs can yield significantly better performance than the popular DNN. The improved performance (about 6-9% relative error reduction) has been observed on two ASR tasks, namely the TIMIT phone recognition dataset and a large vocabulary voice search dataset. Moreover, a set of experiments has been conducted to investigate the effects of various CNN parameters and design settings. Results show that energy information is very beneficial in the CNN.

Moreover, the ASR performance is sensitive to pooling size and it improves a lot as the pooling size increases up to 6. The results also show that an overlap between pooling units is not needed, which leads to better efficiency in storage and computation. Finally, pre-training of CNNs based on the convolutional RBM yields better performance in the large vocabulary voice search dataset.

Chapter 5

Fast Adaptation of DNNs for ASR

Chapter 4 discussed one method of handling the variations in the speech signal by using the CNN. A CNN helps by computing features that are more robust to some speaker variations. This chapter discusses a different method for handling these variations: speaker adaptation.

5.1 Speaker Adaptation in ASR

Speaker adaptation techniques try to optimize system performance towards one target speaker (or a group of speakers). This is done by either modifying the model parameters to match the target speaker or modifying the target speaker's features to match the model. Adaptation depends on collecting a limited amount of adaptation data from each speaker. In most applications, it is desirable to achieve better performance with only a small amount of adaptation data. In supervised adaptation, the labels of the adaptation utterances are assumed to be known. In unsupervised adaptation, the labels are unknown beforehand, and they are either not required or are automatically estimated using a speaker independent ASR system. From the perspective

of the ASR system user, it is more desirable to use unsupervised adaptation; however, unsupervised adaptation is more difficult and more prone to errors.

A lot of research work has been done in speaker adaptation of GMM/HMM models. One popular method is maximum a posteriori (MAP) speaker adaptation [115, 116], in which all HMM parameters are re-estimated to optimize the model for the target speaker. MAP adaptation achieves good performance when a large amount of adaptation data is available. In contrast, maximum likelihood linear regression (MLLR) [117, 118] and constrained MLLR (CMLLR) [119] work much better when only a small amount of adaptation data is used [120]. In these methods, all trained HMM parameters are transformed by a linear transform function that is learned from adaptation data. In CMLLR, this transform can also be viewed as transforming the data itself to match the HMM model. In VTLN [121], a parametric frequency warping function is used to normalize speech features among different speakers. This method is successful and robust since only one free parameter needs to be optimized per speaker. However, its performance is dependent on a manually designed frequency warping function.

5.2 Related Works: NN Adaptation

Recently, the hybrid NN-HMM model has gained more research interest in speech recognition because it has achieved very good performance in both small tasks like TIMIT [90] and large vocabulary tasks [42, 91, 92, 93] especially when a deep NN (DNN) is used. However, speaker adaptation for DNN-HMM models is still a challenging task and it has not received enough attention yet. Some of the adaptation techniques that have been developed for the GMM-HMM can be used with the NN-HMM model if they only transform speech features towards the target speaker without modifying the HMM model. [87, 93] show improvement by using various conventional adaptation techniques like HLDA [122], VTLN, and fMLLR that were developed for the

GMM-HMM models. Though, using DNNs for speech recognition opens the door for different adaptation techniques that are developed especially for the DNN-HMM models and can exploit the potential of the DNN.

A number of attempts have been made to adapt hybrid NN-HMM models. The simplest method is to modify all NN weights using the available adaptation data based on the standard back-propagation (BP) training procedure [123]. But this method is very prone to over-fitting especially when some class labels do not appear in the adaptation data. Another more successful method is to add a linear input network (LIN) to transform the NN input as in [123] (also in [124] named as a transformation network and in [125] an earlier simpler version is presented). During adaptation, only those weights of this linear layer are learned based on the adaptation data. This reduces the number of adaptation parameters and hence reduces the over-fitting problem to some extent. This method is similar to the regression methods that transform speech features with a linear transform. Though, speaker variations are more complex than linear changes. In the linear hidden network method (LHN) [126], the linear transformation layer is inserted above the hidden layers, to transform the features computed by these layers. This can generate a similar effect to non-linear transformation of the speech features. Similarly, the NN outputs are transformed in [127] before computing the normalized softmax output in a method called linear output network (LON). Instead of inserting new layers, a parallel hidden network (PHN) can be added as in [123] where a small number of hidden units are inserted and connect the input and output layers through weights to be learned from the adaptation data. This method reduces the number of trainable parameters compared to LIN and LHN. In [128], a parametric NN activation function is adapted instead of the NN weights. In [129], an affine transformation is used to transform features computed by different DNN layers or the input.

In the previous methods a relatively large number of NN weights need to be estimated from the adaptation data which is relatively small. To re-

duce over-fitting, a number of techniques have been proposed. For example, conservative training in [126] helps to handle missing classes in the adaptation data. Another form of conservative adaptation is proposed in [130] by using a Kullback-Leibler divergence (KLD) regularization which is added to the adaptation criterion. In [131], weight interpolation method is proposed, which is similar to the MAP adaptation in using the SI model as a prior of the new adapted parameters. The effect of L2 regularization (weight decay), and momentum on the adaptation generalization is studied in [132]. Moreover, some constraints on the adapted weights can be imposed to reduce the total number of free parameters to learn from the adaptation data. [127] compares the performance of using a number of structural constraints. The structural constraints aim at discarding or tying some weights based on the domain expert knowledge. In [133], data driven constraints are imposed by using a set of bases to represent the adapted parameters. These bases are estimated using principal component analysis (PCA). During adaptation, the adapted model parameters are estimated by only estimating a small set of weights to estimate the new parameters as a sum of weighted bases.

Most of the above mentioned NN adaptation methods use discriminative training. [127] compares a number of discriminative adaptation techniques, including LIN and LON, to the generative feature space CMLLR technique. It shows that the discriminative techniques generally perform better in supervised adaptation when the correct target labels are available. For unsupervised adaptation, the generative techniques like CMLLR are more robust unless special tricks are performed to reduce the overfitting problem when discriminative techniques are used.

Despite all the previous work on NN-HMM adaptation, fast adaptation, where only a few (from 1 to 10) adaptation utterances are available, is still a challenging problem. In this chapter, we focus on performing fast adaptation and we propose using two new adaptation methods: speaker code based adaptation and adaptive feature scaling weights. These methods aim

at performing fast speaker adaptation while trying to avoid the over-fitting problem. This is achieved by using a relatively small speaker code that can largely reduce the number of adaptation parameters [134, 135, 136] or using scaling weights that are attached to the hidden units' outputs instead of the connections, thus using a much smaller number of weights [137].

5.3 Speaker Code Based Adaptation

The methods described in the previous section perform speaker adaptation by either modifying the model to match the new speaker or learning a new transformation of the new speaker features. In this section we present a different method, that estimates a speaker representation which can be used by the NN to do a better speech recognition exploiting the knowledge of the speaker without modifying the NN itself. The posterior probability of the frame class label depends on information other than the acoustic features of the current frame. For example, in the hybrid DNN-HMM model, it has been found that the performance of classifying the current frame improves when using a wider context window that includes a few frames before and after the current frame, i.e. computing $\mathbf{p}(q_t|\mathbf{X}_{t-r}^{t+r})$. Another kind of information that can help classifying a frame is the speaker ID. In this section we will represent the speaker c using a speaker code $s^{(c)}$ and estimate the probability: $\mathbf{p}(q_t|\mathbf{X}_{t-r}^{t+r}, s^{(c)})$.

An early trial of using a speaker representation within the hybrid NN-HMM model for the simpler task of isolated word recognition has been proposed in [138]. They proposed using a low dimensional speaker representation that is fed to the hidden layer of a NN that has only one hidden layer. In supervised adaptation, the speaker representation can be estimated directly using the error back-propagation algorithm. Their method can be used in unsupervised adaptation as well by integrating over selected points of the speaker space while maintaining the assumption that the utterance

(or group of utterances) come(s) from the same speaker. In [139], hidden factors are used in the top layer to represent speaker and environment in a factorial NN model. The probability distribution of frame labels and these factors given the feature vector can be estimated from the model. The label posterior probability is estimated by marginalizing over all possible factor values but this is done for each frame separately. The work in [140, 141] represent the speaker using i-vectors which are computed directly from the acoustic signal and supplied to the DNN. This method does not require any adaptation utterances.

In this section, we propose performing speaker adaptation of the hybrid DNN-HMM model using a speaker code that represents the speaker features. This speaker code is fed to the DNN along with the frame features. Given the speaker code, the DNN can perform better classification of the given speech frame because it can learn the relation between the speaker features, received speech features, and the target class labels. Two methods are proposed here for feeding this speaker code: direct speaker code based adaptation by feeding the speaker code directly to all or some of the DNN layers through weighted connections, or by using a transformation NN to transform the speech features before feeding them to a speaker independent NN.

To benefit from the speaker code, the DNN needs to learn speaker variations and how they are represented using the speaker code from the training speakers. During adaptation, only the speaker code of the new speaker needs to be estimated. Since speaker variations are learned, the DNN can generalize to new speakers faster, allowing it to perform fast speaker adaptation.

5.3.1 Model Space Speaker Adaptation mSA-SC

Model Structure

In this model, the speaker code is fed as an extra input to all layers of the DNN (or some of them). Fig. 5.1 shows the structure of the model. It shows

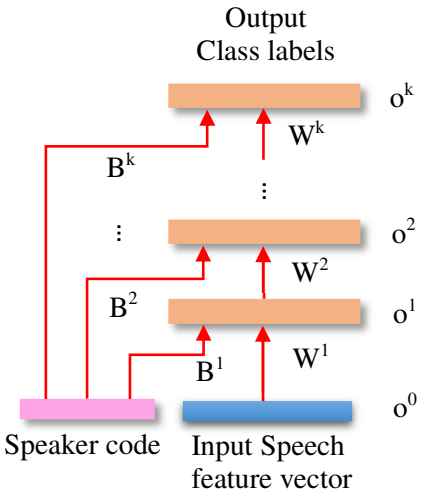


Figure 5.1: Model space - speaker code based adaptation. The figure shows the model structure for a DNN with $k+1$ layers, where the speaker code is fed to all hidden and output layers.

that the model has two sets of weights: the DNN weights $\mathbf{W}^l$ and the speaker code weights $\mathbf{B}^l$. The DNN weights connect the nodes of the lower layer $l-1$ with the current layer l. The speaker code weights connect the speaker code with the current layer l. The activations of the lth layer are computed as follows (as compared to Eq. 2.4):

$$\mathbf{o}^{(l)} = \sigma(\mathbf{o}^{(l-1)}\mathbf{W}^{(l)} + s^{(c)}\mathbf{B}^{(l)}) \qquad (l = 1, 2, \cdots, L-1) \tag{5.1}$$

As can be deduced from Eq. 5.1, the speaker code results in a fixed bias added to the input received from the lower layer. Since the DNN has multiple layers and different biases are added to all of them (depending on the speaker code and the adaptation weights attached to each layer), this can allow the DNN to handle non-linear speaker variations. Since this model is changing the model's hidden layer activations, we will call it *Model Space - Speaker Adaptation Based on Speaker Code* (mSA-SC).

The speaker code is a real vector. The size of the vector is freely specified based on the amount of the adaptation data. Each speaker has a different

speaker code. The speaker code is learned from the speaker adaptation data. The DNN weights and the adaptation weights are fixed among all speakers and are not modified during adaptation. The adaptation weights are learned from all training speakers along with their speaker codes during the training phase. This results in adaptation weights that can learn to handle speaker variations based on the given speaker code. The adaptation weights transform the speaker code into biases that lead to improved performance for the target speaker.

Model Training

During training, the DNN weights $\mathbf{W}^l$ and the adaptation weights $\mathbf{B}^l$ are learned and shared among all speakers. Moreover, a speaker code $s^{(c)}$ is learned for each different speaker in the training set. All these parameters are learned using the same error back-propagation algorithm.

Assuming that we have a trained speaker independent model along with the DNN weights $\mathbf{W}^l, 1 \leq l \leq L$, the adaptation weights $\mathbf{B}^l \in R^{S \times |\mathbf{o}^l|}$ are added and initialized randomly, where S is the speaker code size. Additionally, each speaker in the training data set has a different speaker code $s^{(c)} \in R^S$ that's initialized randomly.

Training proceeds using the stochastic gradient descent (SGD) algorithm, where each mini-batch is used to compute the gradient and update both the speaker code and the adaptation weights. The derivative of the objective function $\mathcal{F}$ with respect to the k-th element of the speaker code $s_k^{(c)}$ can be computed as:

$$\frac{\partial \mathcal{F}}{\partial s_k^{(c)}} = \sum_l \sum_j \frac{\partial \mathcal{F}}{\partial o_j^l} o_j^l (1 - o_j^l) B_{kj}^l = \sum_l \sum_j e_j^l B_{kj}^l, \tag{5.2}$$

where e_j^l is the j-th element of the error vector of the l-th layer $\mathbf{e}^l$, as defined in Eq. 2.8 and Eq. 2.9.

Similarly, the derivative with respect to the weight B_{kj}^l that connects the

k-th element of the speaker code with the j-th node in the l-th layer can be computed as:

$$\frac{\partial \mathcal{F}}{B_{kj}^l} = \frac{\partial \mathcal{F}}{\partial o_j^l} o_j^l (1 - o_j^l) s_k^{(c)} = e_j^l s_k^{(c)} \tag{5.3}$$

A number of strategies can be used to handle the DNN weights $\mathbf{W}^l$. The first strategy is to first learn all DNN weights without any speaker labels and before adding the adaptation weights, and then keeping them fixed during learning of the adaptation weights. A second strategy is to tune them during learning of the adaptation weights. A third strategy is to learn them together with the adaptation weights, starting from a random initialization. The second and third strategies represent a form of speaker adaptive training, where the DNN weights are learned given the speaker IDs and their codes. An empirical comparison of the three training strategies is provided in the experiments section.

Instead of using the minimum cross entropy criterion for training the adaptation weights, an MMI criterion can be used (or any other sequence based discriminative criterion). In MMI, the weights are modified to optimize the recognition accuracy (instead of frame classification accuracy). As will be shown in the experiments section, using MMI improves the overall performance.

Model Adaptation

The model adapts to a new speaker by learning a speaker code for the new speaker. The new speaker code is learned using the same error back-propagation procedure to update the speaker code without changing any weights. The adaptation is performed using a few adaptation utterances that have a known label sequence assuming a supervised adaptation. The frame classification targets are estimated using a forced alignment procedure to align the target label sequence with the speech frames. This can be done using a speaker independent model. After finding each frame class label,

a cross entropy criterion can be minimized by adjusting the speaker code. After the speaker code is learned, it can be used to recognize new utterances from the same speaker.

Other objective functions can be used instead of cross entropy. In [136], it has been shown that using an MMI criterion improves the adaptation performance.

Advantages of the model

The speaker code based adaptation proposed in this section is appealing because adaptation weights which constitute most of the adaptation related parameters are learned from all training speakers, while only a relatively small speaker code is learned for each speaker. This enables rapid speaker adaptation. The adaptation weights learn to handle speaker variations represented using the speaker codes. On the other hand, the speaker representation is learned automatically within the same training procedure. This allows the model to learn other arbitrary variations rather than the speaker as well, given that they are well represented and labeled in the training data. Moreover, a complex non-linear relation between the features, speaker, and class labels can be modeled within the overall DNN and adaptation weights.

5.3.2 Feature Space Speaker Adaptation fSA-SC

Instead of modifying the activations of the DNN layers, we propose to modify the features themselves (i.e. the DNN input) without explicit modification of the hidden layer activations. This can be done by adding an adaptation NN that transforms speech features into a more speaker independent space. By using a multi-layer adaptation NN, this transformation can be non-linear. Thus, it can handle complex variations between different speakers. The transformed features are given as an input to the original speaker independent DNN (possibly after tuning). Moreover, since each speaker has

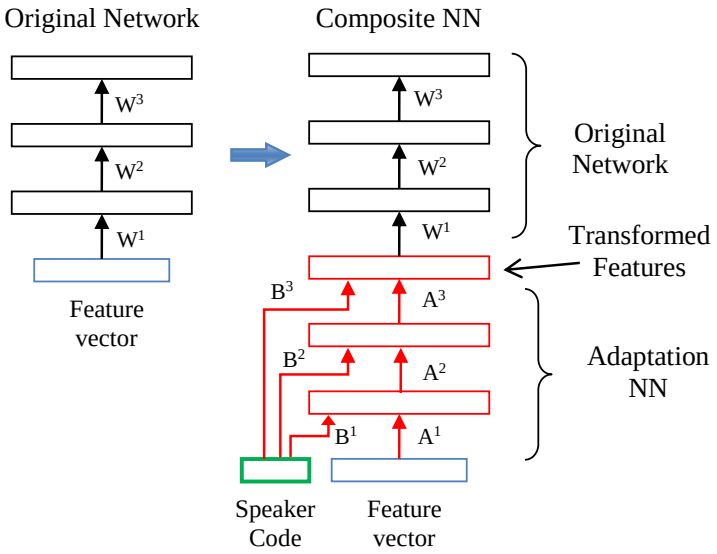


Figure 5.2: Feature transformation - speaker code based adaptation. The figure shows the speaker independent model on the left and the speaker adapted model after adding the adaptation NN on the right.

different features, the speaker code can be fed to the adaptation NN to control the performed transformation. As a result, the adaptation NN can normalize some speaker related variations and present the speech features in a better form for phonetic classification. We will call this method *Feature Space Speaker Adaptation Based on Speaker Code*, denoted as fSA-SC.

The structure of the model is depicted in Fig. 5.2. The figure shows two additional sets of adaptation weights. The adaptation NN weights $\mathbf{A}^l$ and the speaker code weights $\mathbf{B}^l$ (which have been used in the previous section). The adaptation NN weights connect consecutive adaptation NN layers. The speaker code weights connect the speaker code only to the nodes of the adaptation NN. The original speaker independent DNN is kept the same. The only change is that it takes the transformed input instead of the original

input. The activations of the adaptation NN layers can be computed as:

$$\hat{\mathbf{o}}^{(\hat{l})} = \sigma(\hat{\mathbf{o}}^{(\hat{l}-1)}\mathbf{A}^{(\hat{l})} + s^{(c)}\mathbf{B}^{(\hat{l})}) \qquad (\hat{l} = 1, 2, \cdots, \hat{L}), \tag{5.4}$$

where $\hat{L}$ is the number of layers of the adaptation NN (excluding the input layer).

The output layer (the $\hat{L}$-th layer) of the adaptation NN has the same size as the input. To make the output of the adaptation NN more similar to the input a linear activation function can be used in the output layer. An empirical comparison of sigmoid and linear activation functions in the output layer is presented in the experiments section.

Learning of the adaptation weights is like in the mSA-SC model except that we have two sets of weights. Similar to Eq. 5.2 the speaker code can be updated as:

$$\frac{\partial \mathcal{F}}{\partial s_k^{(c)}} = \sum_{\hat{l}=1}^{\hat{L}} \sum_j \frac{\partial \mathcal{F}}{\partial \hat{o}_j^{\hat{l}}} \hat{o}_j^{\hat{l}}(1 - \hat{o}_j^{\hat{l}}) B_{kj}^{\hat{l}} = \sum_{\hat{l}} \sum_j \hat{e}_j^{\hat{l}} B_{kj}^{\hat{l}}, \tag{5.5}$$

where $\hat{e}_j^{\hat{l}}$ is j-th element of the error vector of the $\hat{l}$-th layer of the adaptation NN.

The derivative with respect to the weight $B_{kj}^{\hat{l}}$ that connects the k-th element of the speaker code with the j-th node in the $\hat{l}$-th layer can be computed as:

$$\frac{\partial \mathcal{F}}{B_{kj}^{\hat{l}}} = \frac{\partial \mathcal{F}}{\partial \hat{o}_j^{\hat{l}}} \hat{o}_j^{\hat{l}}(1 - \hat{o}_j^{\hat{l}}) s_k^{(c)} = \hat{e}_j^{\hat{l}} s_k^{(c)} \tag{5.6}$$

Similarly, the derivative with respect to the weight $A_{kj}^{\hat{l}}$ that connects the k-th node of the $(\hat{l}-1)$-th layer to the j-th node of the $\hat{l}$-th layer can be computed as:

$$\frac{\partial \mathcal{F}}{A_{kj}^{\hat{l}}} = \hat{e}_j^{\hat{l}} \hat{o}_k^{\hat{l}-1} \tag{5.7}$$

The original speaker independent DNN weights can be handled in dif-

ferent ways. Either they can be learned completely before learning of the adaptation weights, or they can be tuned or learned from random initialization together with the adaptation weights. Moreover, a good strategy may be to tune only the weights of the first layer to cope with the transformed input. The experiments section shows a comparison between all of these strategies.

During adaptation, only the speaker code of the new speaker needs to be learned. All adaptation weights are kept fixed and shared among all speakers.

In both the mSA-SC and fSA-SC methods, the speaker code size is a design parameter, and it is relatively small. This allows rapid speaker adaptation using only a few adaptation utterances. The fSA-SC method has the advantage of applying all modifications to the features independently of the classification model. For example, the speaker independent DNN can be improved without the need to modify the adaptation NN nor the learned speaker codes. On the other hand, the mSA-SC method has the advantage of direct connection between the DNN nodes and the speaker code. This make learning of the adaptation weights and the speaker codes easier since there are only a fewer layers between the output and the speaker code. In contrast, for the fSA-SC method the error signal has to be back-propagated through multiple layers before reaching the adaptation weights or the speaker code.

5.3.3 Speaker Adaptation of CNNs

Although convolutional neural networks (CNNs) try to normalize speaker variations by adding invariance to small shifts along frequency, explicit speaker adaptation may bring further benefits. The same speaker code based adaptation can be applied to CNNs as well. However, for the fSA-SC method, experiments have shown it is difficult to train the adaptation NN beneath the pooling and convolution layers. A small change in the structure of the fSA-SC method is proposed to overcome this difficulty.

Fig. 5.3 shows the modified structure. It shows that the adaptation NN

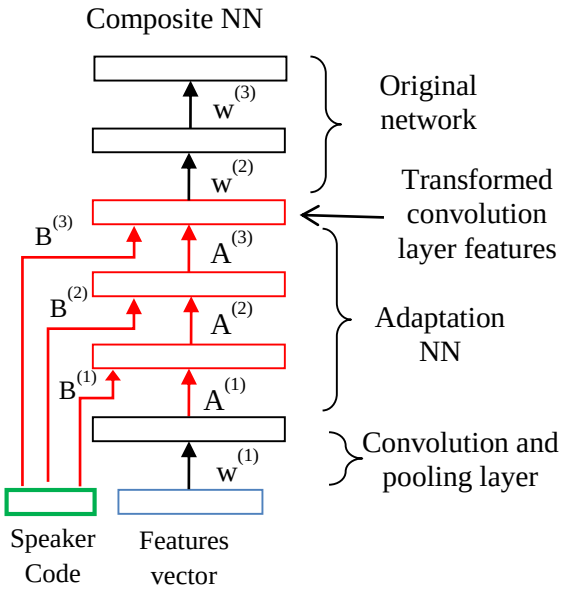


Figure 5.3: Modified feature space speaker code based adaptation for CNNs.

transforms the features generated by the CNN layer instead of transforming the input features. This normalizes other speaker variations that are not handled by the CNN layer, but it makes the adaptation NN dependent on the features computed by the CNN layer.

The CNN cannot handle all forms of speaker variations. That is because complete handling of these variations requires information that exists outside the bounds of the input context window that is fed to the CNN. This extra information is given to the adaptation NN in the form of a speaker code. Given the speaker code, the adaptation NN can normalize more speaker variations and improve the overall performance.

Another possibility is to use a CNN as the adaptation NN to adapt a standard speaker independent DNN. The CNN layers can benefit from the speaker code while they also perform speaker normalization through convolution and pooling. In this case, the CNN serves as an independent speaker normalization system that transforms the speech features into a better form

that is more suitable for classification by any model.

5.4 Adaptive Feature Scaling Weights

A different method to adjust a small number of parameters for speaker adaptation is to add adaptive feature scaling weights. These adaptive weights may further help in adapting DNN or CNN models towards the target speaker. These weights scale either the output generated by a NN node after applying the activation function as shown in Fig. 5.4 or the input features. These weights can be viewed as an extra linear layer with a diagonal weight matrix. These weights can be applied to the features computed by any layer of the DNN or the input layer. In the case of standard fully connected layers, these weights simply increase or decrease the activities of different nodes from layer to layer. In the case of convolution layers, these adaptive weights may serve as a mechanism to force the pooling operations towards certain more preferable frequency shifts. For example, for a male speaker there may be a tendency to decrease activations of nodes representing shifts towards higher frequencies of the same feature map kernels. This may adaptively decrease the effective pooling size, and hence reduce confusion between similar phonemes that sometimes happen when using the CNN [107].

Let's assume that the ith node in a certain NN layer has the output o_i. The scaled output (after multiplying by the scaling weight, v_i) is:

$$\hat{o}_i = o_i \exp(v_i) \tag{5.8}$$

Note that the scaling weight is represented as an exponential of the parameter v_i to guarantee that it is positive during training. The derivative $\frac{\partial E}{\partial \hat{o}_i}$ will be the same as the standard derivative computed using the back-propagation algorithm. But, when the scaling weight is used, the derivative of o_i and v_i

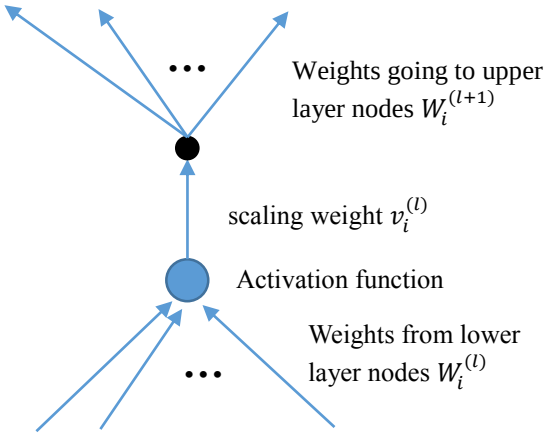


Figure 5.4: Feature scaling weights. The figure shows a scaling weight applied to a hidden node output.

can be computed using:

$$\frac{\partial E}{\partial o_i} = \frac{\partial E}{\partial \hat{o}_i} \exp(v_i) \tag{5.9}$$

and

$$\frac{\partial E}{\partial v_i} = \frac{\partial E}{\partial \hat{o}_i} o_i \exp(v_i) \tag{5.10}$$

For a convolution layer the output of the ith feature map of the jth band at the sth shift location after applying the scaling weight is:

$$\hat{o}_{i,j,s} = o_{i,j,s} \exp(v_{i,j,s}) \tag{5.11}$$

And after applying max pooling it becomes:

$$p_{i,j} = \max_s \hat{o}_{i,j,s} = \max_s o_{i,j,s} \exp(v_{i,j,s}) \tag{5.12}$$

Eq. 5.12 shows that if the weight $\exp(v_{i,j,s})$ is very small for some s, this sth shift will not have any significant effect on the maximization operation. This will force the maximization operation towards certain shifts depending

on the speaker characteristics. Similarly, the scaling weights can be applied after the pooling layer as well.

The introduced scaling weights will be optimized using the speaker adaptation sentences in the same way as the speaker codes in the previous section, except that there is no need to use an adaptation NN. The weight parameters v_i will be initialized with zero then optimized using a number of stochastic gradient descent epochs. Obviously, both the speaker code based adaptation and feature scaling weights adaptation can be combined, where both the speaker code and the scaling weights are jointly learned based on adaptation data in the adaptation stage.

5.5 Experiments

In this section, an experimental evaluation of the proposed adaptation methods is provided. First, the small scale TIMIT phone recognition task is used to compare different methods using the frame level CE training criterion. Afterwards, experiments using the large scale Switchboard task are conducted to measure the performance on large scale large vocabulary speech recognition tasks. Moreover, the MMI sequence training criterion is used and evaluated on the Switchboard task.

5.5.1 TIMIT Phone Recognition

The experiments in this section are performed on the TIMIT phone recognition data set to evaluate the performance of the proposed adaptation techniques.

The experimental setup used here is the same as used in section 4.3, except that user IDs are used to train the adaptation weights and the speaker codes. The same training procedures are used to obtain the speaker independent models.

The training data set has 462 speakers. The development set has 50 speakers, which is used to control the progress of the learning rate and the number of training and adaptation epochs. Adaptation performance results are reported on the core test set which has 24 speakers. Each speaker in all data sets has 8 utterances (after removing the two SA utterances for each speaker).

For training the weights of the original NN and the adaptation NN, the same learning rate annealing and early stopping strategies as in section 4.3 are used. During adaptation, fixed learning rates of 0.1 and 0.025 are used for sigmoid layers and linear layers respectively. The number of epochs is determined using the development set and it is optimized independently for each adaptation data set size. Since each test speaker has eight utterances in total, testing is conducted for each speaker based on a cross validation method. In each run, for each speaker, eight utterances are divided into n_a utterances for adaptation and the remaining $8 - n_a$ utterances for testing. This is repeated eight times for each speaker. Each time, different adaptation and test utterances are randomly selected in such a way that each utterance is assigned the same number of times for both adaptation and testing. The overall recognition performance is the average of all eight runs.

Performance of Different Sizes of the Adaptation Set

In the first set of experiments, the performance of the proposed fast adaptation method is measured using different amounts of adaptation data. In this experiment, a baseline NN with two hidden layers is first trained. The fSA-SC adaptation is used. The adaptation NN has two hidden layers in addition to the output layer, which has a linear activation function. All hidden layers have 1000 nodes each. A speaker code size of 50 is used. The bottom layer of the baseline NN is fine-tuned during learning of the adaptation NN.

Fig. 5.5 shows the adaptation performance using different numbers of adaptation utterances (varying n_a from 0 to 7). With only one adaptation

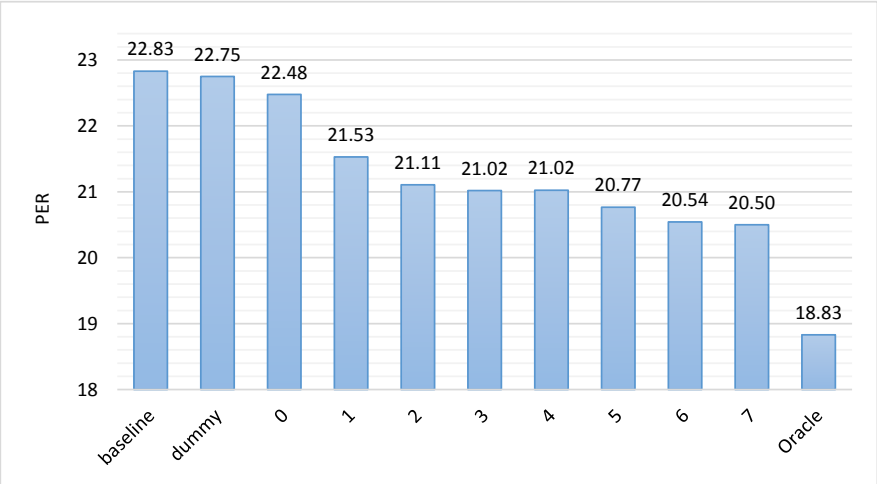


Figure 5.5: Adaptation performance of fSA-SC adaptation (in percent phone error rate) as a function of the number of adaptation utterances.

utterance, the phone error rate is reduced by 1.3% (5.7% relative). After using seven utterances for adaptation, phone error rate (PER) drops from 22.83% to 20.5%. The "dummy" adaptation means that an adaptation NN is added without feeding any speaker codes. The results show that adding these dummy layers achieves only a minor improvement. This confirms that the gain is obtained actually from adaptation not from just adding more layers. Using zero adaptation utterances means that the speaker code is not tuned during adaptation and left as zeros. It indicates that this kind of adaptive training even without adaptation during testing may be a little bit helpful. The oracle score is the PER when the same eight utterances per speaker are used for both adaptation and testing.

Fine-Tuning of the Speaker Independent NN

Here, different schemes of fine-tuning the original speaker independent NN are compared. Since the original NN was trained on the features before adaptation, using it with the adapted features may limit the benefit of the transformed features. As a result, fine-tuning the original NN may improve the

performance. To compare different fine tuning schemes, the same structure as the previous section is used here. Table 5.1 shows the different fine-tuning schemes. It shows that fine tuning only the first layer results in the best performance. Fine-tuning the whole network either from random weights or starting from the baseline NN weights is worse than fine tuning only the first layer. There are a number of possible reasons for this result. One reason is that fine tuning the whole NN may cause faster over-fitting since there are more parameters to tune. Another possible explanation is that the higher DNN layers receive a stronger error signal; thus, the higher layers could fit the input features before the lower layers adjust the adaptation weights and update the speaker code. Further investigation may be required to test the validity of these explanations.

The other observation is that fine-tuning the first layer of the original NN is better than fixing all of its weights during the learning of the adaptation weights. Though, both approaches improve performance. This indicates that the adaptation NN learns to generate different enhanced features that requires changing the original NN to cope with them.

The results shown in this section are only for the fSA-SC adaptation. For the mSA-SC adaption, different results may be obtained. That's because in the mSA-SC model, the speaker code is directly connected to the top layers of the DNN. This may balance the learning of the speaker code and their connecting weights with the DNN weights. The results for the mSA-SC are shown for the switchboard data set.

Linear vs Sigmoid Output Layers of the Adaptation NN

The aim of the adaptation NN is to transform the input features from the speaker space into a speaker-independent space that is more suitable for recognition. This section compares using linear and sigmoid activation functions for the adaptation network's top layer, with and without fine-tuning of the first layer of the original NN. The results in Table 5.2 show that a

Table 5.1: Comparison of different fine-tuning schemes.

Fine tuning scheme	PER
No fine tuning	21.24%
Fine tune first layer	20.50%
Fine tune the whole NN	21.34%
randomly initialized NN	21.48%

Table 5.2: PER (in %) of different activation functions used in the top layer of the adaptation NN. Results are shown for 1 and 7 adaptation utterances.

Activation function	1 utt.	7 utt.
Linear	21.75%	21.24%
Sigmoid	22.21%	21.61%
Linear + fine tune first layer	21.53%	20.50%
Sigmoid + fine tune first layer	21.65%	20.91%

linear top layer yields a slightly better performance. This is because the linear layer can generate features in the same domain as the original features. Hence these features will better suit the original NN weights.

The table also shows that fine tuning the first layer of the original NN is much better because it reduces the discrepancy between the transformed features and the original NN weights.

Speaker Code based Speaker Adaptation of CNNs

This set of experiments evaluates the performance of the speaker code based adaptation method with the adaptation NN re-configuration as proposed in section 5.3.3. The baseline speaker independent CNN has one pair of convolution and pooling layers and two fully connected hidden layers. The CNN uses limited weight sharing and has the following parameters: filter size of

Table 5.3: PER (Phone error rate in %) of CNN adaptation based on speaker code (fSA-SC). The first column shows whether the adaptation NN is added above the convolution layer (Reordering) or not (No reordering), and whether the convolution layer receives the speaker code (SC) or not. The "FT" column shows the index of the fine tuned layers of the original CNN. "1" refers to the convolution layer, and "2" refers to the second hidden layer of original speaker independent CNN.

#	Configuration	FT	PER
1	CNN Baseline	None	20.07%
2	No reordering	1	24.57%
3	Reordering	1	**19.14**%
4	Reordering + CNN gets SC	1	19.25%
5	Reordering + CNN gets SC	2	19.44%
6	Reordering + CNN gets SC	1, 2	19.55%
7	Reordering + CNN gets SC	None	19.31%

8, pooling size of 6, and pooling shift of 2, and it uses 84 feature maps. These experiments test different design and tuning options. The results are listed in Table 5.3, where rows 3 and 4 compare the performance of feeding the speaker code to the convolution layer, and rows 4 to 7 compare different fine-tuning options for the original CNN weights. The table shows that adding the adaptation NN before the CNN layer worsens the performance. Meanwhile, using the adaptation NN to transform the features computed after the pooling layer yields a significant performance gain, namely improving the PER from 20.07% to 19.14% by using only 7 adaptation utterances. The table shows that no gain is obtained by feeding the speaker code to the convolution layer.

Performance of Speaker Code Based Speaker Adaptation with Different Model Architectures

This section compares the improvement of using the proposed adaptation techniques with different baseline model architectures. Both fSA-SC and

mSA-SC are compared with both a relatively small NN with two hidden layers and a deeper NN with five hidden layers. All fully connected layers have 1000 hidden nodes each. Moreover, the performance with CNNs is measured. The CNN structure used is the same as the one used in the experiments in the previous section. The speaker codes sizes are 50 for the fSA-SC model and 300 for the mSA-SC model.

Table 5.4 shows the results. The first three rows show that feature space adaptation works better with the shallower model, while the rows 4 to 7 show that the model space adaptation works better with the deeper model. This can be attributed to the difficulty of learning the adaptation NN when there are many layers separating the output layer and the adaptation NN as the training signal becomes weaker. Moreover, row 7 shows that using a deeper adaptation NN doesn't help. On the other hand, model space adaptation benefits from having deeper models, as this allows the adaptation weights to learn better transformations where the relative reduction of PER increased from 5.4% to 6.2% even with using an improved baseline speaker independent model as in row 5.

The use of a CNN improves the performance either when it's used as the speaker independent model or when it is used as an adaptation NN. Rows 8 and 10 show that adapting a speaker independent CNN using an adaptation NN with fully connected layers achieves better overall performance than using a CNN for adaptation. The best overall performance is obtained in row 10 with a PER of 19.14%.

Performance of Using Adaptive Feature Scaling Weights

This section contains experiments that evaluate the performance of the speaker adaptation using adaptive feature scaling weights as described in section 5.4. Table 5.5 shows the results of the first set of experiments which compare the performance of adding the scaling weights to different layers of a CNN. The same CNN structure is used as in the previous experiment where one pair of

Table 5.4: Comparison of different adaptation model structures. The table shows the performance of adapting three different models: a DNN with 2 hidden layers in rows 1-3, a DNN with 5 hidden layers in rows 4-8, and a CNN in rows 9-10. The third column shows the PER. The fourth column shows the relative reduction in PER as compared with the speaker independent baseline.

	Model	PER	rel. error reduction
1	NN (2×1000)	22.8%	-
2	(1) + mSA-SC	21.6%	5.4%
3	(1) + fSA-SC (2×1000)	20.5%	10.2%
4	NN (5×1000) + PreTraining	21.61%	-
5	(4) + mSA-SC	20.3%	6.2%
6	(4) + fSA-SC (2×1000)	20.70%	4.2%
7	(4) + fSA-SC (4×1000)	20.73%	4.1%
8	(4) + CNN	19.77%	8.5%
9	CNN	20.07%	-
10	(9) + fSA-SC (2×1000)	**19.14%**	4.6%

convolution and pooling layers is used in addition to two fully connected hidden layers. The second column mentions the indexes of the layers to which scaling weights are added, where 0 indicates multiplying the input features by adaptive scaling weights, and 1 indicates scaling the features computed by the convolution layer. The table shows that adaptively scaling the input features is not helpful while adding the scaling weights to layers 1 and 2 yields the biggest improvement (from 20.07% for the baseline to 19.2% after adaptation).

The second set of experiments aims at measuring the performance of combining the two adaptation methods of speaker code and scaling weights. The results are shown in table 5.6. The experiments consider adapting both a DNN with 2 hidden layers and the same CNN structure as in the first set of experiments.

Table 5.5: Adaptation performance based on adaptive scaling weights, which are applied to various layers of the baseline CNN. The second column indicates to which layers the speaker code added. The "layer 0" indicates multiplying the input features by the scalar weights directly.

#	Configuration	PER
0	baseline	20.07%
1	layer 0	20.28%
2	layer 0+1	19.79%
3	layer 0+1+2	19.65%
4	layer 1	19.64%
5	layer 1+2	**19.20**%
6	layer 2	19.21%

For the DNN, the table shows an improvement of using the scaling weights. However the speaker code based method alone yields a better performance, even better than combining the scaling weights method with it. While for the CNN, the scaling weights works equally well as the speaker code method, and combining the two methods brings even more improvement and reaches a PER of 18.86%. These results show that the scaling weights provide an alternative rapid adaptation method that can work with both DNNs and CNNs without the need of training special adaptation weights. Though, the performance is not as good as the speaker code based method. This can be attributed to the lack of generic adaptation weights that can learn a generic transformation function and benefit from the training speakers.

Furthermore, statistical significance tests are performed to confirm the improvement obtained by using the proposed speaker adaptation techniques. Similar to section 4.3, a matched pairs segment sentence word error rate test [114] is performed. The tests show that all adaptation techniques presented in table 5.6 show significant improvement over the the corresponding unadapted model at level $p = 0.001$. Though, the tests on TIMIT do not show significant differences between different adapted models that belong to the same row of the table.

Table 5.6: Adaptation performance by combining and the speaker code based adaptation (SC) and the adaptive scaling weights (SW) methods for both DNN and CNN.

Model	Baseline	SC	OW	SC+OW
DNN	22.83	20.47	21.56	20.69
CNN	20.07	19.14	19.20	**18.86**

5.5.2 Switchboard Experiments

Experimental Setup

In order to confirm the obtained results on TIMIT for the speaker code based adaptation, a set of LVASR experiments have been conducted in [136] on the Switchboard dataset (SWB). The SWB data set consists of the 309 hour Switchboard-I set and the 20 hour Call Home English set, which contain about 1540 speakers in total. In this work, the standard NIST 2000 Hub5e set is used for evaluation which contains 1831 utterances from 40 speakers different from the speakers in the training data set.

For all SWB experiments, PLP features are used to represent the speech signal including the static, first and second derivatives. The speech features are pre-processed with cepstral mean and variance normalization (CMVN) per conversation side. A baseline GMM-HMM with 8,991 tied states and 40 Gaussians per state is first trained based on maximum likelihood estimation (MLE) and then discriminatively trained using the minimum phone error (MPE) criterion. A trigram language model (LM) is trained using 3M words of the training transcripts and 11M words of the "Fisher English Part 1" transcripts. The baseline triphone GMM-HMM model is used to obtain the state level alignment labels for both training and evaluation sets for DNN training and adaptation. The baseline DNNs are trained as described in [93] with the RBM-based pre-training and the BP-based fine-tuning using the frame-level CE criterion.

In the evaluation set (Hub5e00), each test speaker has a different number

of utterances. Two different cross-validation (CV) configurations for adaptation and testing are used. In the first configuration, in each CV run, a fixed number of utterances (10 or 20) are used as adaptation data and the remaining utterances from the same speaker are used to evaluate recognition performance. The process is rotated for many runs until all test utterances are covered. The overall recognition performance is computed as the average of all runs. In the second configuration, the maximum number of utterances per speaker are used for adaptation, called max adaptation. For every test utterance in Hub5e00, all remaining utterances from the same speaker are used to adapt the DNN. Then, the adapted model is used to test this single utterance. The process is repeated for all utterances in Hub5e00. Since the number of utterances is different for each speaker in the test set, the number of adaptation utterances used in this case varies from minimum 25 utterances to maximum 67 utterances per speaker (46 utterances per speaker on average). During adaptation, a fixed learning rate of 0.02 is used to update the speaker code and using a mini-batch of 128 samples. Five epochs are used to learn the speaker codes.

The adaptation NN used in the fSA-SC model has two hidden layers with 2048 nodes each. The speaker code size is 500. To learn the weights of the adaptation NN, a learning rate of 0.1 is used with a learning rate annealing and stopping criterion as the previous section. A learning momentum of 0.9 is used. The used mini-batch size is 1024 samples.

The mSA-SC model uses a similar training procedure, but with a different learning rate schedule. The initial learning rate is 0.5, and it is halved on every epoch after the initial three ones. A bigger code size of 1000 is used due to the lack of the adaptation NN.

Results

The first set of results, obtained by adapting a baseline DNN model that has three hidden layers with 1024 nodes per layer, is shown in table 5.7. It shows

Table 5.7: Performance of speaker code based adaptation in WER % on SWB. The baseline model is a DNN with 3 hidden layers of 1024 nodes each. The relative reduction in WER to the baseline is shown between brackets.

Baseline	WER	Adapt.	10	20	max
DNN (3 × 1024)	18.9%	fSA-SC	17.5	17.3	17.0 (10.0%)
		mSA-SC	17.8	17.8	17.4 (7.9%)

that the fSA-SC model decreases the word error rate (WER) from 18.9% to 17.5% using only 10 adaptation utterances, and to 17.0% using the *max* setup. The mSA-SC model achieves slightly smaller gains with this baseline. On the other hand, the first two rows of table 5.8 show a better performance of the mSA-SC model than the fSA-SC model when a deeper and better baseline DNN is used that has six hidden layers with 2048 nodes each.

The other rows of table 5.8 show more gains by using a number of enhancements to the training and adaptation procedures using the mSA-SC model. Row 3 shows the result of using better label alignments that are obtained using the trained baseline DNN used in row 1. The new alignments are used to train a new speaker independent DNN as a new baseline and adapting it. The new baseline performance improves from 16.2% to 15.9%. After speaker adaptation the WER reaches 14.2%.

Row 4 shows that the baseline significantly improves when a MMI sequence based criterion is used to learn the DNN weights and reaches an error rate of 14.0% and of 13.1% after speaker adaptation. When the MMI criterion is used also to learn the adaptation weights, the WER drops to 12.8% as shown in row 5. In this case, the MMI is used to optimize the weights after the initial weights learned using the CE criterion.

The last two rows, 6 and 7, show the performance when both the adaptation weights $\mathbf{B}^l$ and the DNN weights $\mathbf{W}^l$ are jointly trained in a speaker adaptive training scheme (SAT). To train the SAT models, first the DNN weights $\mathbf{W}^l$ are initialized using the RBM based pre-training. Moreover, the adaptation weights $\mathbf{B}^l$ are initialized with the values obtained in the mSA-

Table 5.8: Performance of speaker code based adaptation with improved optimization in WER % on SWB. The baseline model is a DNN with 6 hidden layers of 2048 nodes each. The "Baseline" column shows the training criterion of the baseline model. The "Adapt" column shows the adaptation model and its training criterion. The last three columns shows the PER % when 10, 20, or the maximum number of adaptation utterances are used.

	Baseline	WER	Adapt.	10	20	max
1	CE	16.2	fSA-SC	15.7	15.5	15.4 (4.9%)
2			mSA-SC	15.2	15.2	14.9 (8.0%)
3	CE-Realigned	15.9	mSA-SC	15.0	14.7	14.2 (10.7%)
4	MMI	14.0	mSA-SC	13.6	13.3	13.1 (6.4%)
5			+ MMI	13.4	13.2	12.8 (8.6%)
6	RBM pre-trained	-	mSA-SC + SAT	13.9	13.5	13.3
7			+ SAT + MMI	12.6	12.4	**12.1**

SC model used to obtain the results in row 5 of the table. Row 6 shows the performance when the whole model is optimized using the CE criterion. When an MMI criterion is used, SAT training achieves the best performance of 12.1% as shown in row 7.

5.6 Conclusion

This chapter has proposed two speaker adaptation methods, a speaker code based speaker adaptation and speaker adaptation using adaptive scaling weights. Both methods are very effective in performing rapid speaker adaptation due to the reduced number of parameters. Experimental results show that both methods improve the performance of the speaker independent baseline and that the speaker code based method achieves more improvement.

Two forms of the speaker code based method have been proposed. The first form performs a model space adaptation where the hidden layer activations of the DNN are modified based on additional adaptation weights that scale the speaker code values reaching each hidden node. The second

form performs a feature space adaptation. This is achieved by adding an adaptation NN that non-linearly transforms the input based on the given speaker code. The original speaker independent DNN (possibly after tuning) receives the transformed input. Experimental results show that the feature space adaptation performs better with shallower DNNs while the model space adaptation performs better with deeper DNNs. A possible interpretation is that the adaptation NN has the capacity to better model speaker variations, but that learning becomes more difficult with deeper models because of the increased number of intermediate layers through which the error signal has to be backpropagated. On the other hand, the model space adaptation contains more weights as more layers are added and hence more learning capacity.

The speaker code method is appealing because of its use of two sets of parameters: a large set of adaptation weights that are learned from all training speakers, and a small set of speaker code parameters that are learned for each speaker separately. The adaptation weights can learn the general variation patterns among different speakers, and thus the personal speaker characteristics can be represented using a small speaker code. This leads to the ability to perform rapid speaker adaptation.

Experimental results show that various modifications can be applied during training of the weights to obtain more improvements in the performance. Results show that obtaining better label alignments using a well trained DNN improves the adaptation performance. Moreover, using an MMI sequence training criterion improves both the speaker independent and adapted models' performance. Finally, they show that a speaker adaptive training strategy improves the system performance especially when an MMI criterion is used. Moreover, the speaker code based method is able to improve the performance of the better performing CNN models.

In addition, the results show that a simple scaling of the DNN features by using adaptive scaling weights can improve the speech recognition performance. This method can be applied to both DNN and CNN models. More-

over, this method can be combined with the speaker code based method to improve the overall system performance.

Finally, the results show that even though a CNN can normalize some speaker variations, more improvements can be obtained when the CNN is combined with speaker adaptation techniques.

Chapter 6

Deep Segmental Neural Network for ASR

6.1 Segmental Models of ASR

All ASR models described in the previous chapters are frame based models which depend on scoring each frame as belonging to one of a number of target classes. Each frame score is computed based on a fixed size input. The final recognition is done by finding the label sequence that has the maximum total score of all frames. The total score is estimated using the HMM, where it combines frame scores to estimate the total score of a certain hypothesis. Typically, the frame scores are the log likelihoods of seeing the observations related to the frame being scored. The score of the whole sequence is estimated by summing the scores of individual frames. However, this summation makes the assumption that consecutive frames are independent of each other given the HMM states, which is unrealistic because consecutive frames are highly correlated especially within the same acoustic unit. Other well known limitations of the HMM model include the restriction of using frame level features and weak duration modeling. Moreover, the DNN used in the hybrid DNN-HMM model is used to compute class label posterior probability given

a fixed size context around the frame. This probability is used to estimate the likelihood of the observation given the state label. Although it works well in practice, this architecture deviates more from the frame conditional independence assumptions of the HMM, since this estimation depends on a wider context, most of which is shared between consecutive frames. Moreover, this conversion from state (senone) label posterior probability to conditional likelihood may lead to modeling difficulties. All these problems of the HMM and frame-based models may impair the performance of these models.

An alternative that has received some attention is to use speech segmental models [142, 143]. However, segmental models are more complex and they have achieved less success than frame-based ones. A segmental model estimates scores for whole segments directly. Hence, it handles the dependency between frames in the segment in a better way. Moreover, it can use different kinds of features that depend on the knowledge of segment boundaries including segment duration. However, segmental models are more computationally complex to use since all possible segment sizes may have to be considered during decoding. Moreover, the model should be able to compare labels of different lengths that map to different numbers of observations in the form of segments. For some models, this requires using some form of normalization of scores for sequences of different lengths.

One class of segmental models is generative. Generative segmental models try to model the likelihood of generating variable length segments given the segment labels. The HMM is a special case of these models when the segment frames are conditionally independent. For the sake of comparison, the HMM computes the likelihood of an observation sequence y_1^T given the label sequence a_1^N as:

$$\mathbf{p}(y_1^T|a_1^N) = \sum_{s_1^T} \mathbf{p}(y_1^T|s_1^T, a_1^T)p(s_1^T|a_1^N) \tag{6.1}$$

where the sum ranges over all state sequences s_1^T that are possible for the

label sequence a_1^N. The observation likelihood given the state sequence can be estimated using:

$$\mathbf{p}(y_1^T|s_1^T, a_1^T) = \prod_{t=1}^{T} \mathbf{p}(y_t|s_t) \tag{6.2}$$

In the HMM, the conditional likelihood of a single frame $\mathbf{p}(y_t|s_t)$ can be estimated from a GMM or by using an NN in the hybrid model. The factor $p(s_1^T|a_1^N)$ designates the possible pronunciations and transition probabilities between the HMM states which implicitly models the duration as a geometric distribution. For a generative segmental model, all frames in each utterance are grouped into a number of segments. To estimate the likelihood of an observation sequence, all possible segment lengths, represented as the sequence l_1^N, should be considered. The likelihood can be formulated as described in [142]:

$$\mathbf{p}(y_1^T|a_1^N) = \sum_{l_1^N} \mathbf{p}(y_1^T|l_1^N, a_1^T)\mathbf{p}(l_1^N|a_1^N), \tag{6.3}$$

where

$$\mathbf{p}(y_1^T|l_1^N, a_1^T) = \prod_{i=1}^{N} \mathbf{p}(y_{t(i-1)+1}^{t(i)}|l_i, a_i), \tag{6.4}$$

where $y_{t(i-1)+1}^{t(i)}$ is the sequence of frames that belong to the ith segment. Eq. 6.3 allows the use of an explicit duration model represented as the probability $\mathbf{p}(l_1^N|a_1^N)$.

Different models are possible for modeling the segment frames and their trajectory within the segment. Some techniques try to model the segment trajectory either as a parameteric model as in [144] and [143] or as a non-parametric model as in [145, 146, 147]. Other work modeled the relation between consecutive frames using a conditional Gaussian distribution where a frame is conditioned on the previous one as done by Wellekens [148]. The work in [149] used a linear dynamical system while the work in [150, 151] used a nonlinear dynamical system. [142] presented further discussion of

different models used for modeling the segment observations. Note that all these efforts depend on frame-based features.

Other models use variable length features that are called landmarks [152]. A generative probabilistic framework is used to model both segment features and the landmarks. Moreover, it handles the difference of length between different label sequence hypotheses by using anti-phones for normalizing the resultant different score spaces.

Another class of segmental models depends on direct classification models. The *segmental conditional random field* (SCRF) model has received much interest as a successful direct model [153, 154, 155]. It models the label sequence posterior probability directly without using the Bayes' rule. The SCRF uses feature functions that measure the relation between acoustic and linguistic events and segment labels. The overall posterior probability of the segment label sequence is modeled as a log linear model of the values of these functions. The learning in this model is done by finding the optimal weights that are multiplied with the values of these feature functions. This model avoids the restrictive conditional frame independence assumption that is imposed by the HMM. Moreover, it allows using different kinds of features that are not restricted to single frames and it may benefit from the information of segment boundaries. Two disadvantages of the SCRF are that it depends on a rich set of hand designed features and is a shallow model.

This chapter proposes a new segmental model called a *deep segmental neural network* (DSNN) [156]. This model combines both the deep neural network and the segmental direct model. A deep neural network computes segment scores for each possible segmentation and label combination given the frame observations of this variable length segment. The DSNN better exploits the output of the DNNs without the need of indirect computation of observation likelihoods from the posteriors estimated from the DNN. It directly exploits the scores computed from the DNNs. Moreover, this model does not require any manual tuning of the features, since they are learned

directly within the DNN. This chapter first describes the formulation of the model and how segment scores are computed. Then, it describes how learning of model parameters is performed. The experimental setup and obtained results are described next. At the end, conclusions are drawn.

6.2 The Deep Segmental Neural Network

6.2.1 Model formulation

The model depends on using a neural network for estimating scores for different labels for variable length segments. These scores are used to estimate the posterior probability of the whole label sequence. Assuming we are given a sequence of feature vectors, X, for an utterance, we use $L = \{l_1, \cdots, l_K\}$ to represent a sequence of labels, which may be defined at the subphoneme, phoneme, syllable or even word level, and $T = \{t_0, t_1, \cdots, t_K\}$ to denote one particular time alignment for the label sequence, i.e. the boundaries of segments. The label sequence and the associated time sequence form a segment sequence. The conditional probability for the segment sequence Y given the speech utterance X is estimated as

$$P(L,T|X) = \frac{\exp\left(\sum_i s(l_i, t_{i-1}+1, t_i|X) + u(L)\right)}{\sum_{\hat{L},\hat{T}} \exp\left(\sum_j s(\hat{l}_j, \hat{t}_{j-1}+1, \hat{t}_j|X) + u(\hat{L})\right)}, \tag{6.5}$$

where $s(l_i, t_{i-1}+1, t_i|X)$ represents the acoustic score of getting label l_i for the segment that has the time boundaries $[t_{i-1}+1, t_i]$, and $u(L)$ stands for the total LM score computed for the entire label sequence L. The denominator in Eq. 6.5 sums over all possible label sequences $\hat{L}$ and time alignments $\hat{T}$. If we are only interested in the label sequence L, we can sum over all possible time alignments to yield the posterior probability of one particular L given

X as:

$$\begin{aligned} P(L|X) &= \sum_{T} P(L,T|X) \\ &= \frac{\sum_T \exp\left(\sum_i s(l_i, t_{i-1}+1, t_i|X) + u(L)\right)}{\sum_{\hat{L},\hat{T}} \exp\left(\sum_j s(\hat{l}_j, \hat{t}_{j-1}+1, \hat{t}_j|X) + u(\hat{L})\right)}. \end{aligned} \tag{6.6}$$

where the time alignment T is considered as a hidden variable and it is handled by summing over all possible time alignments for a given label sequence. Another possibility is to consider only the best time alignment as valid, i.e. maximizing over different time alignment as follows:

$$P(L|X) = \frac{\max_T \exp\left(\sum_i s(l_i, t_{i-1}+1, t_i|X) + u(L)\right)}{\sum_{\hat{L}} \max_{\hat{T}} \exp\left(\sum_j s(\hat{l}_j, \hat{t}_{j-1}+1, \hat{t}_j|X) + u(\hat{L})\right)}. \tag{6.7}$$

In this work, the first formulation in Eq. 6.6 is called the *sum formulation* (SF) while the second formulation in Eq. 6.7 is called the *max formulation* (MF). The max formulation is more similar to what is done during decoding where only the best time alignment is found. The experiments compare the performance of the two formulations.

In the proposed model, a DNN is used to compute the acoustic scores for each segment, $s(l_i, t_{i-1}+1, t_i|X)$, and thus the model is named the *deep segmental neural network (DSNN)*. The scores used here may take values of any suitable range and they do not need to be log probabilities. The total acoustic and language score of a label and segmentation sequence is the negated value of the energy function of the model. The DNN learns to compute the scores that maximize the conditional probability of the label of the training data. This can be done by using a linear activation function in the output layer. The probability of classifying the segment with a certain label increases as the score of this label increases and vice versa.

Typically, the DNN receives a fixed size input. Hence, it is challenging to feed the segment boundary information and handle the variable length

feature vectors of the segment. In [157], a segmental neural net model was proposed, where the variable length segment was sampled to a fixed number of frames and some frames may be skipped or repeated. In this work, we propose two methods to compute the segment scores. In the first method the segment score is computed from a frame-based DNN that receives a fixed size input and computes a score for each frame of the segment. Afterwards, a simple aggregation function is used to combine the variable number of scores and estimates a single score that represents the whole segment. In the second method, a more complicated and powerful segment aware DNN is used to compute the whole segment score. The two methods are described in the following sections.

Note that any type of LM can be used in the above definition. In the performed experiments, a simple phoneme bigram LM is used to compute $u(L)$. Other more complex LMs can be used as well but they may require some approximations such as constraining the search space with word graphs instead of summing over all possible segment sequences. Moreover, a method like that in [153] can be used, where different nodes of the language model representing different n-grams are considered as different states and each segment is classified to one of these states.

6.2.2 Segment score based on frame-based DNNs

A standard DNN takes a fixed length input while a speech segment has a variable length. In this section, the DNN computes label scores for each frame. An aggregation function is used to convert the multiple scores computed for all segment frames into one score for each label that represents the whole segment. Three different aggregation functions are presented here: middle frame, last frame, and summation. The three aggregation functions are shown in Figs. 6.1(a-c). These methods are simple score functions and they do not benefit from the segment boundary information.

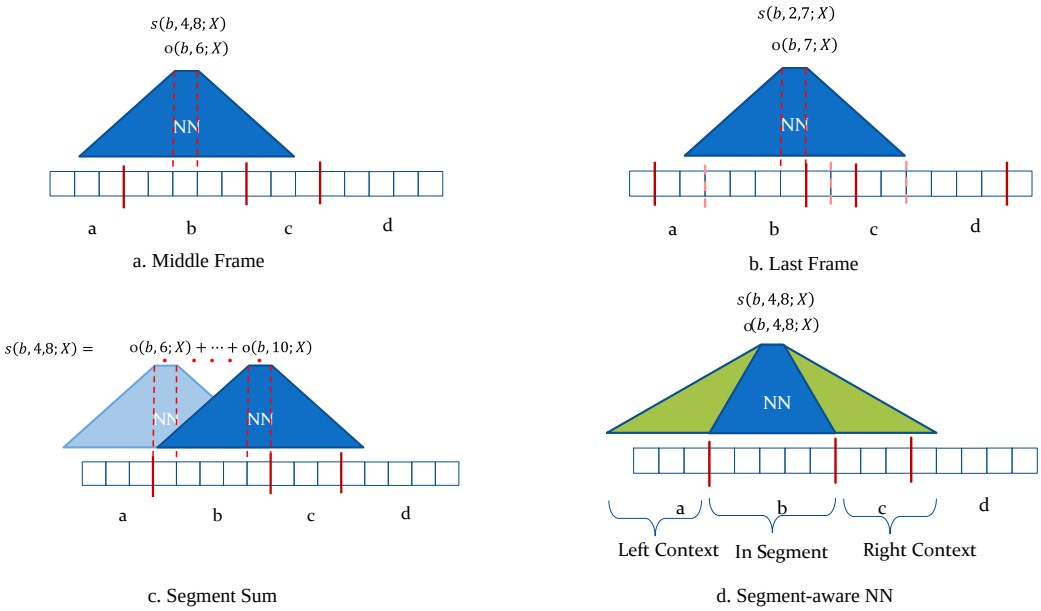


Figure 6.1: Different methods of computing segment scores.

Middle frame

The first method, shown in Fig. 6.1.a, estimates the segment's score using the DNN score computed for the middle frame within the segment; i.e.,

$$s(l_i, t_{i-1}+1, t_i|X) = o(l_i, \frac{t_i + t_{i-1} + 1}{2}) \tag{6.8}$$

This method requires that the DNN receives a large enough fixed length input that covers all or most of the segment frames. The disadvantage of this method is that it may prefer deleting segment labels if the score is negative, which may happen if a certain segment is noisy or is not clear enough. This leads to increased deletion errors.

Final frame

This method picks an arbitrary frame from the segment as the representative frame and defines the segment boundaries as starting after the last frame of

the previous segment and ending at the selected frame as shown in Fig. 6.1.b. The segment score function is defined as :

$$s(l_i, t_{i-1} + 1, t_i|X) = o(l_i, t_i). \tag{6.9}$$

In this method, the selected segment boundaries are not the true ones. This is not a problem since the aim of speech recognition is finding the label sequence and the segment time alignments are not needed for most applications. This method is more flexible than the previous one and allows the system to pick the frame that has the maximum score; however, it has the same problem as the previous one of increased deletion errors.

Summation of segment frames

This method estimates the segment score as the sum of the scores of all frames in the segment as shown in Fig. 6.1.c. The segment score is defined as:

$$s(l_i, t_{i-1} + 1, t_i|X) = \sum_{t=t_{i-1}+1}^{t_i} o(l_i, t_i) \tag{6.10}$$

The advantage of this method is that it considers all frames in the utterance without skipping any. This reduces the number of deletion errors since if a segment label is deleted, the scores of the frames of this segment will be added to the wrong segment and the overall score is decreased. This makes the system prefer to avoid these deletion errors. On the other hand, this method results in a bigger scale for scores from longer segments, since more scores are summed.

6.2.3 Segment scores based on a segment-aware DNNs

In order to better exploit segment boundary information, a segment aware DNN is used. It takes all segment information and directly computes segment scores. As shown in Fig. 6.1.d, the DNN takes all segment frames in addition

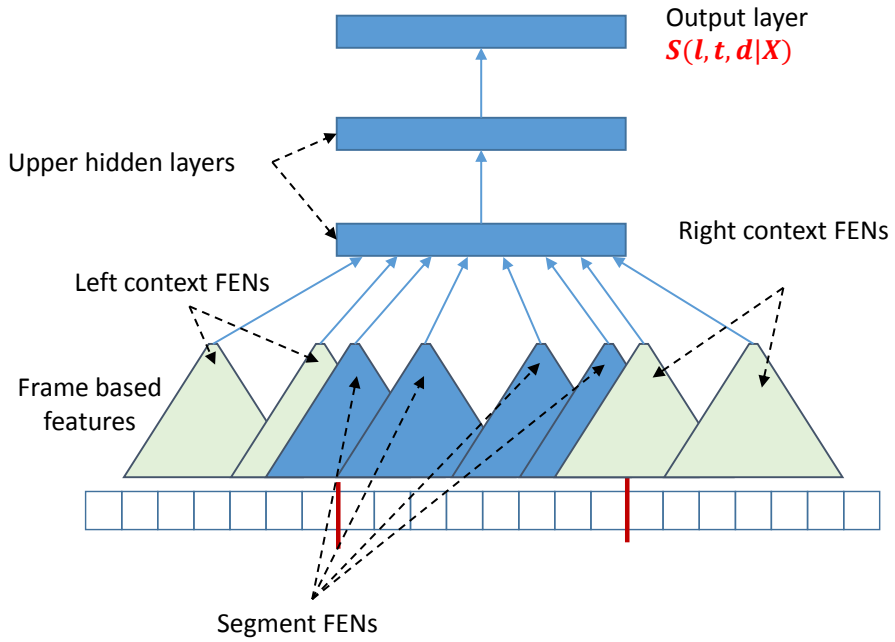


Figure 6.2: Structure of the deep segmental neural network (DSNN)

to a number of context frames. However, speech segments are of variable length and the DNN expects fixed-length inputs. To solve this problem, this section proposes to use the structure shown in Fig. 6.2. Instead of interpolating the segment frames into a fixed number of frames, a number of DNNs, which take a fixed size input, are used to compute features of different parts of the segment. These DNNs are represented as trapezoids in the lower part of Fig. 6.2 and will be called *Feature Extraction Networks* (FENs). Since there are a fixed number of these FENs distributed along the segment, the features computed by them are concatenated as a fixed size feature vector that represents the variable size segment. This feature vector is further analyzed by a number of other upper hidden layers and an output layer that are shown in the upper part of the figure. The output layer computes the scores of assigning different labels to the given segment. The combined upper layers and lower FENs forms a composite DNN that is called a *segment aware DNN*.

In this work, the FENs are distributed along different parts of the segment as follows: N_c FENs are distributed inside the segment boundaries with equal space between them. The first and last of these FENs are placed just after and before the segment boundary (i.e. at time $t_{i-1}+1$ and t_i). N_l FENs process the left context and N_r FENs process the right context and all of them are equally spaced. The space between the FENs that process the segment depends on the segment length, while the other FENs that process the left and right contexts have a fixed space. In Fig. 6.2, the following values are used: $N_c = 4$ and $N_l = N_r = 2$.

The weights of the FENs may be tied. In this work, all FEN weights are tied except the output layer which may or may not be tied to allow the computation of different features.

6.3 Learning of DSNN Model

This section describes the training of the DSNN model for either the sum formulation in Eq. 6.6 or the max formulation in Eq. 6.7. In this work, the label sequence posterior probability in Eqs. 6.6 and 6.7 is maximized. This technique is called conditional maximum likelihood estimation (CMLE) in the ASR community. This is considered as a discriminative training technique where the posterior probability of the correct label sequence is maximized while the posteriors of incorrect label sequences are minimized. Other objective functions are possible, like minimum phone or word error criteria or large margin estimation. The training can either proceed in a stochastic gradient descent method where the NN weights are updated after computing the derivative for each utterance, or in a batch mode by the summing the derivatives of all utterances. In either case, the derivative for one utterance is needed.

6.3.1 Learning based on the sum formulation

In the sum formulation of Eq. 6.6, all time alignments are considered. Assume that X represents the frames of the speech utterance and L is the correct label sequence. For any particular weight matrix, $\mathbf{W}$, in the DSNN, the derivative of logarithm of the objective function in Eq. 6.6 can be computed based on the chain rule as follows:

$$\frac{\partial \ \log p(L|X)}{\partial \ \mathbf{W}} = \sum_{l,t_s,t_e} \frac{\partial \ \log p(L|X)}{\partial \ s(l,t_s,t_e)} \cdot \frac{\partial \ s(l,t_s,t_e)}{\partial \ \mathbf{W}} \tag{6.11}$$

where $s(l,t_s,t_e)$ denotes the segmental acoustic score computed by the DNN defined by $\mathbf{W}$.

The second derivative in the right hand side of Eq. 6.11 can be computed directly using the back-propagation algorithm. The first derivative represents the error signal that reaches the top layer for each input that represents a different segment label or boundaries. It can be computed based on Eq. 6.6 as follows:

$$\frac{\partial \log p(L|X)}{\partial s(l,t_s,t_e)} = \frac{\sum_{T \in \mathcal{A}(L,l,t_s,t_e)} p(L,T|X)}{p(L|X)} - \sum_{(\hat{L},\hat{T}) \in \mathcal{B}(l,t_s,t_e)} p(\hat{L},\hat{T}|X) \tag{6.12}$$

where $\mathcal{A}(L,l,t_s,t_e)$ denotes the set of time alignments of the label sequence L that assign time boundaries (t_s,t_e) with label l, and $\mathcal{B}(l,t_s,t_e)$ denotes the set of all possible label sequences and time alignments that embed the segment (l,t_s,t_e)).

The summations in Eq. 6.12 contain an exponentially increasing number of terms. However, if a bigram language model is used in Eq. 6.6, these summations can be recursively evaluated using the forward-backward algorithm. In this case, the forward backward algorithm is defined slightly differently than for the HMM. $\alpha_s(l,t)$ is defined as the sum of partial scores of all paths that lead to label l starting at time t excluding the current label score. $\alpha_e(l,t)$

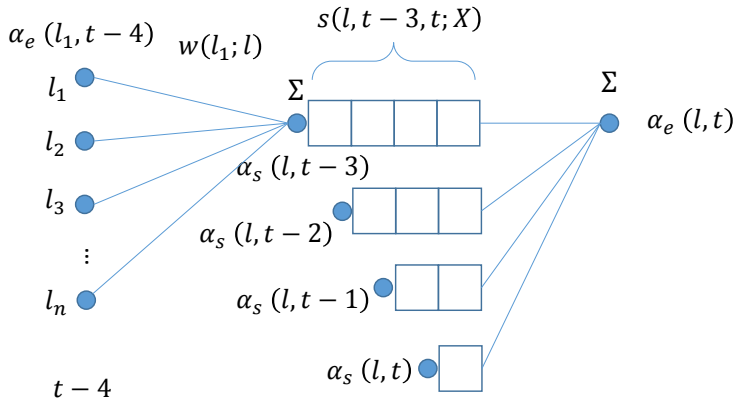


Figure 6.3: Illustration of recursive "forward" computations of α_s and α_e.

is the sum of partial scores of all paths that end with a segment label l at time t. Fig. 6.3 illustrates one step in computing $\alpha_s(l,t)$, which accounts for all labels before time t, and one step in computing $\alpha_e(l,t)$, which considers all different lengths of segment l ending at time t. These two quantities can be computed recursively according to:

$$\alpha_s(l,t) = \sum_{\hat{l}} \alpha_e(\hat{l}, t-1) \exp\left(w(l;\hat{l})\right) \tag{6.13}$$

and

$$\alpha_e(l,t) = \sum_{d} \alpha_s(l, t-d+1) \exp\left(s(l, t-d+1, t)\right) \tag{6.14}$$

where d represents the segment duration that is summed from 1 to the maximum duration of segment l, and $w(l;\hat{l})$ is the language model score for transitioning from label $\hat{l}$ to l.

Similarly, β_s and β_e are defined for the backward direction as:

$$\beta_e(l,t) = \sum_{\hat{l}} \beta_s(\hat{l}, t+1) \exp\left(w(\hat{l};l)\right) \tag{6.15}$$

$$\beta_s(l,t) = \sum_{d} \beta_e(l, t+d-1) \exp\left(s(l, t, t+d-1)\right) \tag{6.16}$$

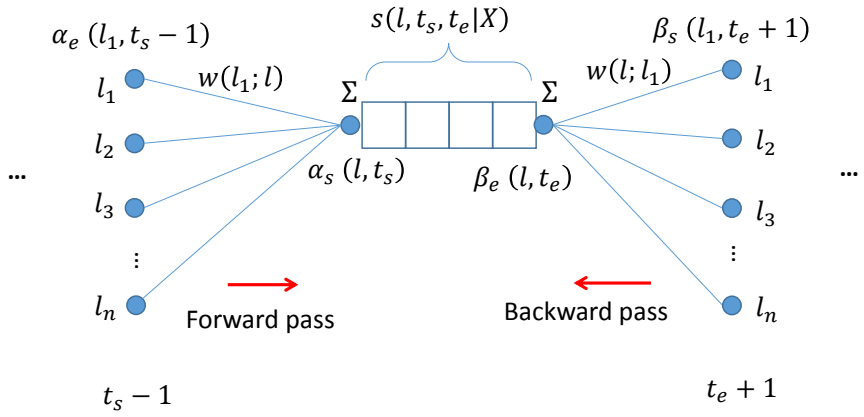


Figure 6.4: Illustration of the summation of score paths that include the segment (l, t_s, t_e).

Based on the above definitions, the terms of Eq. 6.12 can be estimated. The second term can be estimated as:

$$\sum_{(\hat{L},\hat{T})\in\mathcal{B}(l,t_s,t_e)} p(\hat{L}, \hat{T}|X) = \frac{\alpha_s(l, t_s) \exp\big(s(l, t_s, t_e)\big)\beta_e(l, t_e)}{\sum_l \alpha_e(l, |X|)}, \tag{6.17}$$

where the numerator is the sum of all paths that include the segment (l, t_s, t_e) based on α and β computations as shown in Fig. 6.4. The denominator is the total sum of the scores of all labels sequences and time alignments at the last frame $|X|$. It is the sum of all possible endings of the label sequence. Usually, only one label that represents a start/end silence state is allowed for the beginning or the end of the sequence.

To estimate the first term in Eq. 6.12, the label sequence has to be restricted to the correct L. This can be done efficiently by redefining α and β in terms of the index of the label within the correct label sequence instead of the label itself as follows:

$$\hat{\alpha}_s(i, t) = \hat{\alpha}_e(i - 1, t - 1) \exp\big(w(l_i; l_{i-1})\big), \tag{6.18}$$

$$\hat{\alpha}_e(i,t) = \sum_d \hat{\alpha}_s(i, t-d+1) \exp\big(s(l_i, t-d+1, t)\big) \tag{6.19}$$

and:

$$\hat{\beta}_e(i,t) = \hat{\beta}_s(i+1, t+1) \exp\big(w(l_{i+1}; l_i)\big), \tag{6.20}$$

$$\hat{\beta}_s(i,t) = \sum_d \hat{\beta}_e(i, t+d-1) \exp\big(s(l_i, t, t+d-1)\big) \tag{6.21}$$

Hence:

$$\frac{\sum_{T \in A(L,l,t_s,t_e)} p(L,T|X)}{p(L|X)} = \sum_{i|l_i=l} \frac{\hat{\alpha}_s(i,t_s) \exp\big(s(l,t_s,t_e)\big) \hat{\beta}_e(i,t_e)}{\hat{\alpha}_e(|L|,|X|)} \tag{6.22}$$

Model learning requires the computation of $s(l, t_s, t_e)$ for all possible l, t_s, and t_e, where the duration of each label l is limited to the maximum duration seen for each label in the training set. This computation has been efficiently implemented by parallelization in a GPU. After these computations, the derivatives of the log objective function are back-propagated to all DNNs to update each weight matrix via stochastic gradient ascent.

6.3.2 Learning based on the max formulation

In order to learn using the gradient descent method, the derivative of Eq. 6.7 is to be estimated as follows:

$$\frac{\partial \ \log p(L|X)}{\partial \ \mathbf{W}} = \sum_{l,t_s,t_e} \frac{\partial \ \log p(L|X)}{\partial \ s(l,t_s,t_e)} \cdot \frac{\partial \ s(l,t_s,t_e)}{\partial \ \mathbf{W}}, \tag{6.23}$$

which has the same form as Eq. 6.11. But the first derivative considers only the maximum time alignment and is estimated as follows:

$$\frac{\partial \ \log p(L|X)}{\partial \ s(l,t_s,t_e)} = \begin{cases} 1 - \sum\limits_{(\hat{L},\hat{T})\in\mathcal{B}(l,t_s,t_e)} p(\hat{L},\hat{T}|X) & \text{if } (l,t_s,t_e) \in \mathcal{S}(L,T^*) \\ 0 - \sum\limits_{(\hat{L},\hat{T})\in\mathcal{B}(l,t_s,t_e)} p(\hat{L},\hat{T}|X) & \text{if } (l,t_s,t_e) \notin \mathcal{S}(L,T^*) \end{cases} \tag{6.24}$$

where $\mathcal{S}(L,T^*)$ is the set of segments $(l_i, t_{i-1}+1, t_i)$ that are defined by L and T^*, and T^* is the time alignment of L that has the maximum score. Alternatively, T^* can be the correct target alignment if it is available. Note that $\mathcal{B}(l,t_s,t_e)$ is defined differently here and it includes only the label sequences having max time alignments that embed the segment (l,t_s,t_e).

The max formulation depends on finding the best time alignment of the correct label sequence and also of each alternative label sequence to compute the posterior probability in Eq. 6.7. The same is also needed during learning to compute the sum of posterior probabilities in Eq. 6.24. The problem is that there is an exponentially large number of possible label sequences and it is not possible to estimate the scores of all these label sequences in a reasonable time. In practice, however, only a few of these sequences have a large enough score to affect the total posterior probability or to compete with the correct label sequence. Hence most of the label sequences can be pruned in a fairly early stage. A strategy that uses an N-best list of sequences will give a good approximation of the learning signals. Moreover, these competing label sequences will have many shared sub-sequences. An efficient solution is to find a lattice that comprises all competing label sequences and estimate the scores directly from the lattice. This lattice will include most of the N-best sequences.

Fig. 6.5 shows a part of a lattice. The lattice has a number of nodes where each node represents segments that have the same label and end at the same time. The arcs coming into this node are for the same segment label

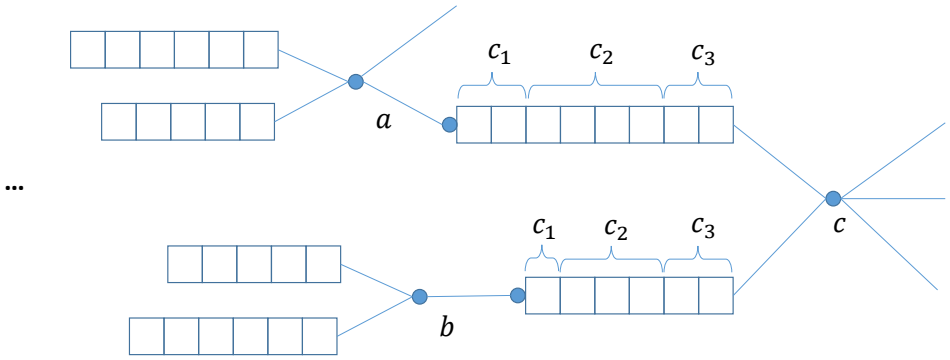


Figure 6.5: Training Lattice with N = 2. It shows two alternative starting times of segments C1,C2, and C3 (that map to phoneme C) each with a different history (Phonemes A and B).

but may have different start times and different label histories. To estimate the posterior probability, the partition function is estimated by summing the scores of all arcs of the lattice. The sums in Eq. 6.24 are estimated by summing the scores of all segments in the partial paths that lead to the segment (l, t_s, t_e) plus the scores of all paths starting at the end of this segment which can be done efficiently with a linear complexity.

6.4 Decoding

In decoding, the aim is to search for the best label and alignment sequence for each speech utterance X in the test set. With the use of a bigram language model, the search can be carried out using the Viterbi version of the forward algorithm in Eqs. 6.13 and 6.14 by replacing summation with maximization as follows:

$$\alpha_s(l,t) = \max_{\hat{l}} \alpha_e(\hat{l}, t-1) \exp\left(w(l;\hat{l})\right) \tag{6.25}$$

$$\alpha_e(l,t) = \max_{d} \alpha_s(l, t-d+1) \exp\left(s(l, t-d+1, t)\right) \tag{6.26}$$

Moreover, in order to be able to recover the best label sequence and time alignments, the max history should be stored during the iterations of the Viterbi algorithm as follows:

$$\theta_s(l,t) = \underset{\hat{l}}{\operatorname{argmax}}\, \alpha_e(\hat{l}, t-1) \exp\left(w(l;\hat{l})\right) \tag{6.27}$$

$$\theta_e(l,t) = \max_{d} \alpha_s(l, t-d+1) \exp\left(s(l, t-d+1, t)\right) \tag{6.28}$$

This decoding is much slower than the standard HMM Viterbi algorithm as it requires consideration of all possible segment durations. In the performed experiments, decoding run time is accelerated by parallelizing the Viterbi search on the CPU cores and the DSNN segment scores computation on the GPU.

6.5 Experimental Evaluation

6.5.1 Experimental setup

Experiments are performed on the standard TIMIT phone recognition task using the core test set. The experimental setup of the baseline hybrid DNN-HMM model is the same as the one used in section 4.3.

The DSNN models 61 phonemes and each phone is modeled as a sequence of three segments. Hence, a segment represents a sub-phonetic unit similar to an HMM state. For training the DSNN model, weight derivatives are computed as described in section 6.3.1 and the weights are updated after processing each utterance. The same weight annealing and stopping strategy is used as the hybrid DNN-HMM model and as described in section 4.3. The SF-DSNN model doesn't need any time alignments since it sums all time alignments for both the correct label sequence and the competing sequences. Target time alignments may be used for training the MF-DSNN

model. The time alignments are estimated from a trained HMM model. In the experiments presented here, time alignments are used during the training of the MF-DSNN model unless mentioned otherwise. The log probabilities of a bigram language model are used as the language model scores in DSNN model unless mentioned otherwise. Duration scores are not used in the DSNN models.

6.5.2 Results

Experiments are conducted to measure the performance of the proposed DSNN with different formulations and score functions. Table 6.1 summarizes the results and compares the DSNN to the hybrid DNN/HMM model. All the models shown in the table have 4 fully connected hidden layers with 1000 nodes. The composite segment aware DNN has eight FENs. Each FEN has one hidden layer with 1000 nodes and the hidden layer weights are shared among the eight FENs. Each FEN has 150 units in the output layer with unshared weights. This leads to 1200 nodes in the second hidden layer of the composite segment aware DNN after concatenating the FEN outputs (just slightly bigger than the 1000 nodes of the used DNNs). The FENs are distributed as described in Fig. 6.2 where four FENs are distributed uniformly within the segment boundaries and two FENs are placed outside each side of the segment with a shift of three frames between them. For example, if a segment starts at time t and ends at time $t + 6$, the context FENs will be placed at times: $t - 4$, $t - 1$, $t + 7$, and $t + 10$, while the segment FENs will be placed at times: t, $t + 2$, $t + 4$, and $t + 6$.

The first row of table 6.1 shows the performance of the hybrid DNN-HMM model. Since all DSNN experiments does not use a duration model, the DNN-HMM model score without using the implicit HMM duration model (represented as state transition probabilities) is presented in the second row. Rows from 3 to 5 show the performance of the SF-DSNN model with different simple aggregation functions when a frame based DNN is used. All of them

have worse PERs than the DNN-HMM model. This may indicate that there are some problems with the proposed method as compared with the DNN-HMM model. The DSNN model training optimizes the sequence posterior probability as opposed with the frame classification cross entropy criterion used in the DNN-HMM model. Another factor is that the mini-batch samples are selected randomly in the case of the DNN-HMM model, while they belong to same utterance in the case of the DSNN model. This randomization of batch samples is proven to be better than using samples coming from the same utterance. On the other hand, the segment aware DNN achieves better performance than the frame based DNNs and achieves a PER of 22.9%. This indicates that the segment-aware DNN can benefit from the segment boundary information and handle the correlation within the segment better than the simple aggregation within the HMM model or by using the shown simple aggregation functions. Row 7 shows that when the max formulation is used with a segment aware DNN using a lattice with size parameter (N) being 16, the performance is significantly improved over either the sum formulation or the hybrid DNN-HMM model. This shows that the max formulation performs better than the sum formulation.

Moreover, statistical significance test is performed to confirm the improvement obtained by using the MF-DSNN model. Similar to section 4.3, a matched pairs segment sentence word error rate test [114] is performed. The test shows that the performance obtained by using the MF-DSNN model shown in row 7 of the table is significantly better than the DNN models in rows one and two at level p = 0.05.

Table 6.2 shows the performance of different structures of the segment-aware DNN when the sum formulation is used. The same FENs distribution is used as in table 6.1. It shows that using deeper structures is better. Moreover, the best performance is achieved when the FEN output weights are not shared. In this case, each FEN is computing a different feature vector suitable to its location within the segment. Moreover, using a CNN

Table 6.1: Performance of the proposed DSNN model with different segment score functions and formulations on the TIMIT data set. The performance is shown in PER% when a language model is used (LM) and without using a language model (no LM).

	Model	LM	no LM
1	Hybrid DNN-HMM (with duration)	22.28%	-
2	Hybrid DNN-HMM (no duration)	23.31%	24.63%
3	SF-DSNN - Middle Frame	25.61%	24.72%
4	SF-DSNN - Last Frame	24.59%	25.36%
5	SF-DSNN - Segment Sum	25.42%	25.35%
6	SF-DSNN - Segment Aware DNN	22.90%	23.92%
7	MF-DSNN - Segment aware DNN	**21.60%**	**22.13%**

layer improves the performance of the model and achieves a PER of 21.87%.

Table 6.3 shows the performance with different numbers of FENs. Generally, increasing the number of FENs increases the performance. On the other hand, increasing the size of the FEN output layer does not improve the performance. Table 6.4 shows that better performance is obtained with deeper models (both the FENs and the upper hidden layers).

Table 6.5 shows the effect of training lattice size on the performance of the trained MF-DSNN model. The first column 'N' defines the lattice size in terms of the number of different histories kept at a phone terminating node. The table shows that a good performance is obtained with N of 4 while about 24 minutes are taken for one training epoch. This is only about five times slower than the hybrid DNN-HMM model while significantly better in performance. Increasing N further prolongs the training time while it just slightly improves the performance.

Table 6.2: PER comparisons among different structures of the composite segment-aware DNN on TIMIT within the SF-DSNN model. The first column shows the number of hidden units in each hidden layer. The two pairs of brackets represent the lower FENs and the upper hidden layers, respectively. The second column shows whether the weights of the FENs output layers are shared or not. The last row shows the performance when the first hidden layer is a LWS-CNN layer (a pair of convolution and pooling layers) that has 84 feature maps and computes 20 frequency bands.

Architecture	features sharing	PER
{300}, {1000}	shared	24.15%
{300}, {1000}	non-shared	24.40%
{1000,500}, {1000,1000}	shared	23.52%
{1000,150}, {1000,1000}	non-shared	22.90%
{CNN,150}, {1000,1000}	non-shared	21.87%

Table 6.3: Effect of using different number of FENs in the segment aware DNN on the performance of the MF-DSNN model. The first column shows the number of FENs inside the segment. The second column shows the number of FENs on the left or right contexts. The third column shows the number of units in the FEN output layer.

MF	LF/RF	FS	Test	Dev
3	1	150	22.2	20.26
3	1	250	22.75	20.38
3	2	150	22.12	20.12
4	1	150	22.8	20.39
4	2	150	21.79	20.03

Table 6.4: Effect of the depth of the segment aware DNN on the performance of the MF-DSNN model. The first two columns show the number of layers of the FENs (DF) and the number of upper hidden layers (DU).

DF	DU	Test	Dev
1	2	22.47	20.48
2	1	22.15	20.08
2	2	21.79	20.03
3	2	21.75	19.53
3	3	**21.26**	**19.47**

Table 6.5: The performance and training time of the MF-DSNN model on TIMIT dataset with different training lattice sizes (N). The first column shows the number of different histories kept at each lattice node. The performance is shown for the core test set (Test) and the development set (Dev) in PER% and the training time in minutes per epoch.

N	Test	Dev	Epoch Time
4	21.79	20.03	24
8	21.79	19.63	33
16	21.60	19.45	44

6.6 Conclusions

This chapter has presented a novel segmental model — the deep segmental neural network. The DSNN estimates the acoustic scores for variable-length segments and models the label sequence conditional probability directly. This eliminates the assumption that consecutive frames are independent of each other given the state and thus it has a potential to perform better than the DNN/HMM hybrid model.

A number of segment scoring functions have been described in this chapter. The segment aware DNN has shown the best performance as compared with the other frame-based ones. This can be attributed to the improved capability to handle the dependency between consecutive frames within the segment when the composite segment aware DNN is used.

Moreover, two mathematical formulations of the label sequence posterior probability have been presented. Experimental results show that the max-formulation outperforms the sum-formulation. The max-formulation maximizes only the best time alignment which is more consistent with decoding. This may be one reason for the better performance of the MF-DSNN model. Another reason may be the difficulty of distributing suitable scores between different time alignments of the same label sequence using the sum formulation. More experiments are needed to prove the validity of these interpretations.

Experimental results show that the bi-gram log probabilities can be used as language scores that lead to improved performance. Other language models can be used as well, but further research is needed to improve language modeling within the DSNN model.

Moreover, the current research did not attempt to directly model the segment duration. Further research is needed to investigate suitable methods to exploit the duration information and incorporate it within the model.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This dissertation has presented a number of novel methods that exploit the power of DNNs to improve the performance of ASR systems.

Chapter 4 has proposed applying CNNs to speech recognition in such a way that CNN structures can explicitly handle certain speech variations. The CNN structure is designed based on the understanding of the nature of speech signals. Invariance to small frequency shifts is obtained by applying convolution and pooling along the frequency axis. These frequency shifts normally occur in speech signals due to speaker differences. In addition, a new limited weight sharing scheme has been proposed which can handle speech features in a better way. Experimental results show that it provides a slightly better accuracy than the standard full weight sharing scheme originally proposed for image recognition applications. Moreover, the limited weight sharing scheme leads to a much smaller number of units in the pooling layer, resulting in a smaller model size and lower computational complexity than the full weight sharing scheme. Experimental results show that CNNs can yield significantly better performance than the popular DNNs. The improved performance (about 6-9% relative error reduction) has been observed

on two ASR tasks, namely the TIMIT phone recognition dataset and a large vocabulary voice search dataset. Moreover, a set of experiments has been conducted to investigate the effects of various CNN parameters and design settings. Results show that (1) energy information is very beneficial in a CNN; (2) ASR performance is sensitive to pooling size, with a large enough pooling size being needed to achieve the best accuracy; and (3) pre-training a CNN using the convolutional RBM can yield better performance.

Chapter 5 has proposed two speaker adaptation methods, one using speaker codes and one using adaptive scaling weights. Both methods can perform rapid speaker adaptation due to the reduced number of free parameters in adaptation. Experimental results show that both methods improve the performance of the speaker independent baseline and that the speaker code based method achieves better improvement.

Two forms of the speaker code based method have been proposed. The first form performs a model space adaptation where the hidden layer activations of the DNN are modified based on additional adaptation weights that scale the speaker code values reaching each hidden node. The second form performs a feature space adaptation. This is achieved by adding an adaptation NN that non-linearly transforms the input based on the given speaker code, and the original speaker independent DNN (possibly after tuning) receives the transformed input. Experimental results show that the feature space adaptation performs better with shallower DNNs while the model space adaptation performs better with deeper DNNs. A possible interpretation is that the adaptation NN has the capacity to better model speaker variations, but that the learning of these variations becomes more difficult with deeper models because of the increased number of intermediate layers through which the error signal has to be back-propagated. Conversely, model space adaptation gains weights as more layers are added and hence more learning capacity.

The speaker code method is appealing because of its use of two sets of pa-

rameters: a large set of adaptation weights that are learned from all training speakers, and a small set of speaker code parameters that are learned for each speaker separately. The adaptation weights can learn the general variation patterns among different speakers; thus, the personal speaker characteristics can be represented using a small speaker code. This enables rapid speaker adaptation.

Experimental results show that certain modifications applied during training of the weights can lead to more improvements in the performance. The results show that obtaining better label alignments using a well trained DNN improves the adaptation performance. Moreover, using an MMI sequence training criterion improves both the speaker independent and adapted models' performance. Finally, they show that a speaker adaptive training strategy improves the system performance especially when an MMI criterion is used. Moreover, the speaker code based method is able to improve the performance of the better performing CNN models.

The results also show that a simple scaling of the DNN features by using adaptive scaling weights can improve the speech recognition performance. This method can be applied to both DNN and CNN models. Moreover, this method can be combined with the speaker code based method to improve the overall system performance.

Chapter 6 has presented a novel segmental model — the deep segmental neural network (DSNN). The DSNN estimates the acoustic scores for variable-length segments and models the label sequence's conditional probability directly. This eliminates the assumption that frames are independent of each other given the state and thus has potential to perform better than the DNN-HMM hybrid.

A number of segment scoring function are described in this chapter. The segment aware DNN shows the best performance as compared to the other frame-based ones. This can be attributed to the improved handling of dependency between consecutive frames within the segment when the composite

segment aware DNN is used.

Moreover, two mathematical formulations of the label sequence posterior probability are proposed. Experimental results show that the max-formulation performs better than the sum-formulation. The max-formulation maximizes only the best time alignment, which is what is done during decoding. This may be one reason for the better performance of the MF-DSNN model. Another reason may be that it is difficult to assign suitable scores to different time alignments of the same label sequence using the sum formulation. More experiments are needed to prove the validity of these interpretations.

Experimental results show that the N-gram log probabilities can be used as language scores and they lead to improved performance. Other language models can be used as well. Though, further research is need to improve language modeling within the DSNN model.

7.2 Future Work

The proposed work in this dissertation may open up some new venues of research that can lead to even better deep neural network models. The following list includes a number of related ideas that need further research:

- Applying convolution and pooling through time proved to be beneficial. Further research is needed to enhance the convolution structure and combine frequency and time convolution. In [158], experimental results showed improved accuracy when a special structure of convolution along time is combined with convolution through frequency. Though, the work does not use pooling along time. I believe that pooling can help in handling speed variations that lead to temporal shifts, but that further research is needed to find suitable structures.

- The CNN is believed to provide more robustness against certain kinds

of noise. Though, further experiments are needed to test the ability of CNN in handling noisy speech as compared with the DNN. The same is to be said to compare the improvement obtained by CNN over DNN with multi-speaker dataset as compared with that of a single speaker dataset.

- Finding the speaker code through back-propagation may need further research. As more iterations may lead to over-fitting, the role of regularization in the estimation of speaker codes should be examined.

- Speaker codes may be extended to account for information beyond speaker characteristics such as environment factors, accent, and other information that can be inferred from the speech context.

- The proposed segmental DSNN has shown improved performance over the DNNs that process fixed length inputs on the TIMIT phone recognition task. Further experiments are needed to confirm the performance improvement on large vocabulary ASR tasks. Moreover, the model can be used to re-score N-best lists or recognition lattices generated by other models, which still needs to be investigated.

- The proposed mathematical formulation of the label sequence posterior probability in the DSNN model allows integration of arbitrary sources of information like language and duration. Conducted experiments showed the benefit of adding language model information. Further research is needed to measure the effect of adding other sources like duration.

- In the conducted experiments, a bigram language model is used but it is not optimized within the DSNN framework. Further research can find better ways to learn language model parameters.

- The proposed DSNN has been used to model the structure of speech features within the segment frames. The same method can be extended

to combine the features of consecutive segments by adding other layers in a hierarchy that process longer spans. Further research in this direction is needed.

Bibliography

[1] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[2] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, Oct. 1986.

[3] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems (MCSS)*, vol. 2, pp. 303–314, 1989.

[4] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, pp. 251–257, March 1991.

[5] I. Sutskever and G. E. Hinton, "Deep, narrow sigmoid belief networks are universal approximators," *Neural Computation*, vol. 20, no. 11, pp. 2629–2636, November 2008.

[6] N. Le Roux and Y. Bengio, "Deep belief networks are compact universal approximators," *Neural Computation*, vol. 22, no. 8, pp. 2192–2207, Aug. 2010.

[7] Y. Bengio, "Learning deep architectures for AI," *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009, also published as a book. Now Publishers, 2009.

[8] Y. Bengio and Y. Lecun, "Scaling learning algorithms towards AI," in *Large-Scale Kernel Machines*, L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, Eds. MIT Press, 2007, pp. 321–359.

[9] D. Erhan, P.-A. Manzagol, Y. Bengio, S. Bengio, and P. Vincent, "The difficulty of training deep architectures and the effect of unsupervised pre-training," in *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS 2009)*, Apr. 2009, pp. 153–160.

[10] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Advances in Neural Information Processing Systems 19 (NIPS'06)*. MIT Press, 2007, pp. 153–160.

[11] G. Tesauro, "Practical issues in temporal difference learning," *Machine Learning*, vol. 8, pp. 257–277, May 1992.

[12] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *International Conference on Artificial Intelligence and Statistics*, 2010, pp. 249–256.

[13] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, November 1998.

[14] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Deep big simple neural nets excel on handwritten digit recognition," *CoRR*, vol. abs/1003.0358, 2010.

[15] G. E. Hinton and S. Osindero, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, p. 2006, 2006.

[16] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio, "An empirical evaluation of deep architectures on problems with many factors of variation," in *Proceedings of the 24th International Conference on Machine Learning (ICML'07)*. ACM, 2007, pp. 473–480.

[17] H. Larochelle and Y. Bengio, "Classification using discriminative restricted Boltzmann machines," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 536–543.

[18] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *Proceedings of the 26th Annual International*

Conference on Machine Learning, ser. ICML '09. New York, NY, USA: ACM, 2009, pp. 609–616.

[19] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504 – 507, July 2006.

[20] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Computation*, vol. 14, no. 8, pp. 1711–1800, 2002.

[21] T. Tieleman, "Training restricted Boltzmann machines using approximations to the likelihood gradient," in *Proceedings of the 25th international conference on Machine learning*, ser. ICML '08. New York, NY, USA: ACM, 2008, pp. 1064–1071.

[22] T. Tieleman and G. Hinton, "Using fast weights to improve persistent contrastive divergence," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ser. ICML '09. New York, NY, USA: ACM, 2009, pp. 1033–1040.

[23] A. U. Asuncion, Q. Liu, A. T. Ihler, and P. Smyth, "Particle filtered MCMC-MLE with connections to contrastive divergence," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. Omnipress, June 2010, pp. 47–54.

[24] M. Welling, M. Rosen-Zvi, and G. Hinton, "Exponential family harmoniums with an application to information retrieval," in *Advances in Neural Information Processing Systems 17*. Cambridge, MA: MIT Press, 2005, pp. 1481–1488.

[25] T. Schmah, G. E. Hinton, S. L. Small, S. Strother, and R. S. Zemel, "Generative versus discriminative training of RBMs for classification of fMRI images," in *Advances in neural information processing systems*, 2008, pp. 1409–1416.

[26] S. Osindero and G. Hinton, "Modeling image patches with a directed hierarchy of Markov random fields," in *Advances in Neural Information Processing Systems 20*. Cambridge, MA: MIT Press, 2008, pp. 1121–1128.

[27] R. Memisevic and G. Hinton, "Unsupervised learning of image transformations," in *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, june 2007, pp. 1 –8.

[28] G. W. Taylor and G. E. Hinton, "Factored conditional restricted Boltzmann machines for modeling motion style," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ser. ICML '09. New York, NY, USA: ACM, 2009, pp. 1025–1032.

[29] M. Ranzato, A. Krizhevsky, and G. E. Hinton, "Factored 3-way restricted Boltzmann machines for modeling natural images." *Journal of Machine Learning Research - Proceedings Track*, vol. 9, pp. 621–628, 2010.

[30] R. Memisevic and G. E. Hinton, "Learning to represent spatial transformations with factored higher-order Boltzmann machines," *Neural Computation*, vol. 22, pp. 1473–1492, 2007.

[31] G. E. Hinton, "Learning to represent visual input," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 365, no. 1537, pp. 177–184, 2010.

[32] R. M. Neal, *Bayesian Learning for Neural Networks*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1996.

[33] M. Ranzato and G. Hinton, "Modeling pixel means and covariances using factorized third-order Boltzmann machines," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, june 2010, pp. 2551 –2558.

[34] M. Ranzato, V. Mnih, and G. E. Hinton, "Generating more realistic images using gated MRF's," in *Advances in Neural Information Processing Systems*, 2010, pp. 2002–2010.

[35] A. Courville, J. Bergstra, and Y. Bengio, "A spike and slab restricted Boltzmann machine," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, ser. JMLR W&CP, vol. 15, Apr. 2011.

[36] ——, "Unsupervised models of images by spike-and-slab RBMs," in *Proceedings of the Twenty-eight International Conference on Machine Learning (ICML'11)*, Jun. 2011.

[37] G. E. Hinton, P. Dayan, B. J. Frey, and R. Neal, "The wake-sleep algorithm for unsupervised neural networks," *Science*, pp. 1158–1161, May 1995.

[38] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" in *Proc. International Conference on Computer Vision (ICCV'09)*. IEEE, 2009.

[39] H. Larochelle, D. Erhan, and P. Vincent, "Deep learning using robust interdependent codes," in *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS 2009)*, Apr. 2009, pp. 312–319.

[40] D. Erhan, A. Courville, Y. Bengio, and P. Vincent, "Why does unsupervised pre-training help deep learning?" in *JMLR W&CP: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, vol. 9, May 2010, pp. 201–208.

[41] A. Mohamed, G. Dahl, and G. Hinton, "Acoustic modeling using deep belief networks," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 1, pp. 14–22, Jan. 2012.

[42] F. Seide, G. Li, and D. Yu, "Conversational speech transcription using context-dependent deep neural networks," in *Proc. INTERSPEECH*, 2011, pp. 437–440.

[43] S. Katz, "Estimation of probabilities from sparse data for the language model component of a speech recognizer," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 35, no. 3, pp. 400–401, Mar 1987.

[44] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent neural network based language model." in *INTERSPEECH*, 2010, pp. 1045–1048.

[45] J. Baker, "The DRAGON system–an overview," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 23, no. 1, pp. 24–29, Feb 1975.

[46] M. Bishop and E. Thompson, "Maximum likelihood alignment of DNA sequences," *Journal of molecular biology*, vol. 190, no. 2, pp. 159–165, 1986.

[47] R. Durbin, *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge university press, 1998.

[48] A. Kundu and P. Bahl, "Recognition of handwritten script: a hidden Markov model based approach," in *Acoustics, Speech, and Signal Processing, 1988. ICASSP-88., 1988 International Conference on*. IEEE, 1988, pp. 928–931.

[49] L. E. Baum, J. Eagon *et al.*, "An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model for ecology," *Bull. Amer. Math. Soc*, vol. 73, no. 3, pp. 360–363, 1967.

[50] L. R. Rabiner and B.-H. Juang, *Fundamentals of speech recognition*. PTR Prentice Hall Englewood Cliffs, 1993, vol. 14.

[51] L. B. Bahl, P. de Souza, and R. P Mercer, "Maximum mutual information estimation of hidden Markov model parameters for speech recognition," in *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'86.*, 1986.

[52] B.-H. Juang, W. Hou, and C.-H. Lee, "Minimum classification error rate methods for speech recognition," *Speech and Audio Processing, IEEE Transactions on*, vol. 5, no. 3, pp. 257–265, 1997.

[53] S. Katagiri, B.-H. Juang, and C.-H. Lee, "Pattern recognition using a family of design algorithms based upon the generalized probabilistic descent method," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2345–2373, 1998.

[54] D. Povey and P. C. Woodland, "Minimum phone error and I-smoothing for improved discriminative training," in *Acoustics, Speech, and Signal*

Processing (ICASSP), 2002 IEEE International Conference on, vol. 1. IEEE, 2002, pp. I–105.

[55] H. Jiang, X. Li, and C. Liu, "Large margin hidden Markov models for speech recognition," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 14, no. 5, pp. 1584–1595, 2006.

[56] X. Li and H. Jiang, "Solving large-margin hidden Markov model estimation via semidefinite programming," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 15, no. 8, pp. 2383–2392, 2007.

[57] Y. Normandin, R. Cardin, and R. De Mori, "High-performance connected digit recognition using maximum mutual information estimation," *Speech and Audio Processing, IEEE Transactions on*, vol. 2, no. 2, pp. 299–311, 1994.

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

[59] R. P. Lippmann and B. Gold, "Neural classifiers useful for speech recognition," in *1st International Conference on Neural Networks, IEEE*, 1987.

[60] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang, "Phoneme recognition using time-delay neural networks," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 37, no. 3, pp. 328–339, Mar 1989.

[61] A. Waibel, "Modular construction of time-delay neural networks for speech recognition," *Neural computation*, vol. 1, no. 1, pp. 39–46, 1989.

[62] A. Waibel, H. Sawai, and K. Shikano, "Modularity and scaling in large phonemic neural networks," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 37, no. 12, pp. 1888–1898, 1989.

[63] M. Gori, Y. Bengio, and R. De Mori, "BPS: A learning algorithm for capturing the dynamic nature of speech," in *Neural Networks, 1989. IJCNN., International Joint Conference on.* IEEE, 1989, pp. 417–423.

[64] P. Haffner, A. Waibel, H. Sawai, and K. Shikano, "Fast back-propagation learning methods for large phonemic neural networks." in *Eurospeech*, 1989, pp. 2553–2556.

[65] T. Robinson and F. Fallside, "A recurrent error propagation network speech recognition system," *Computer Speech & Language*, vol. 5, no. 3, pp. 259–274, 1991.

[66] P. Cosi, P. Frasconi, M. Gori, and N. Griggio, "Phonetic recognition experiments with recurrent neural networks," in *Second International Conference on Spoken Language Processing*, 1992.

[67] H. Bourlard and N. Morgan, "Continuous speech recognition by connectionist statistical methods," *Neural Networks, IEEE Transactions on*, vol. 4, no. 6, pp. 893–909, Nov 1993.

[68] 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*, Apr 1990, pp. 413–416 vol.1.

[69] H. A. Bourlard and N. Morgan, *Connectionist speech recognition: a hybrid approach*. Springer, 1994, vol. 247.

[70] S. Renals, N. Morgan, H. Bourlard, M. Cohen, and H. Franco, "Connectionist probability estimators in HMM speech recognition," *Speech and Audio Processing, IEEE Transactions on*, vol. 2, no. 1, pp. 161–174, 1994.

[71] H. Franco, M. Cohen, N. Morgan, D. Rumelhart, and V. Abrash, "Context-dependent connectionist probability estimation in a hybrid hidden Markov model-neural net speech recognition system," *Computer Speech & Language*, vol. 8, no. 3, pp. 211–222, 1994.

[72] J. Hennebert, C. Ris, H. Bourlard, S. Renals, and N. Morgan, "Estimation of global posteriors and forward-backward training of hybrid HMM/ANN systems." in *EUROSPEECH*, 1997.

[73] Y. Yan, M. Fanty, and R. Cole, "Speech recognition using neural networks with forward-backward probability generated targets," in *Acoustics, Speech, and Signal Processing, IEEE International Conference on*, vol. 4. IEEE Computer Society, 1997, pp. 3241–3241.

[74] A. J. Robinson, G. Cook, D. P. Ellis, E. Fosler-Lussier, S. Renals, and D. Williams, "Connectionist speech recognition of broadcast news," *Speech Communication*, vol. 37, no. 1, pp. 27–45, 2002.

[75] A. J. Robinson, "An application of recurrent nets to phone probability estimation," *Neural Networks, IEEE Transactions on*, vol. 5, no. 2, pp. 298–305, 1994.

[76] D. Kershaw, "Phonetic context-dependency in a hybrid ANN/HMM speech recognition system," Ph.D. dissertation, Cambridge University Engineering Department, 1996.

[77] E. Trentin and M. Gori, "A survey of hybrid ANN/HMM models for automatic speech recognition," *Neurocomputing*, vol. 37, no. 1, pp. 91–126, 2001.

[78] 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.

[79] F. Grezl, M. Karafiát, S. Kontár, and J. Cernockỳ, "Probabilistic and bottle-neck features for LVCSR of meetings." in *ICASSP (4)*, 2007, pp. 757–760.

[80] F. Valente, M. Magimai-Doss, C. Plahl, S. V. Ravuri, and W. Wang, "A comparative large scale study of MLP features for Mandarin ASR." in *INTERSPEECH*, 2010, pp. 2630–2633.

[81] Q. Zhu, A. Stolcke, B. Y. Chen, and N. Morgan, "Using MLP features in SRI's conversational speech recognition system." in *INTERSPEECH*, 2005, pp. 2141–2144.

[82] D. Vergyri, A. Mandal, W. Wang, A. Stolcke, J. Zheng, M. Graciarena, D. Rybach, C. Gollan, R. Schlüter, K. Kirchhoff *et al.*, "Development of the SRI/nightingale Arabic ASR system." in *INTERSPEECH*, 2008, pp. 1437–1440.

[83] G. E. Dahl, M. Ranzato, A. Mohamed, and G. E. Hinton, "Phone recognition with the mean-covariance restricted Boltzmann machine," in *Advances in Neural Information Processing Systems*, ser. NIPS, no. 23, 2010.

[84] A. Mohamed, T. Sainath, G. Dahl, B. Ramabhadran, G. Hinton, and M. Picheny, "Deep belief networks using discriminative features for phone recognition," in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, may 2011, pp. 5060 – 5063.

[85] D. Yu, L. Deng, and G. Dahl, "Roles of pre-training and fine-tuning in context-dependent DBN-HMMs for real-world speech recognition," in *Proc. NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2010.

[86] G. Dahl, D. Yu, L. Deng, and A. Acero, "Large vocabulary continuous speech recognition with context-dependent DBN-HMMs," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2011.

[87] F. Seide, G. Li, X. Chen, and D. Yu, "Feature engineering in context-dependent deep neural networks for conversational speech transcription," in *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, 2011, pp. 24–29.

[88] N. Morgan, "Deep and wide: Multiple layers in automatic speech recognition," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 20, no. 1, pp. 7–13, 2012.

[89] G. E. Hinton, "To recognize shapes, first learn to generate images," *Progress in brain research*, vol. 165, pp. 535–547, 2007.

[90] A. Mohamed, G. Dahl, and G. Hinton, "Deep belief networks for phone recognition," in *NIPS Workshop on Deep Learning for Speech Recognition and Related Applications*, 2009.

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

[92] T. N. Sainath, B. Kingsbury, B. Ramabhadran, P. Fousek, P. Novak, and A. Mohamed, "Making deep belief networks effective for large vocabulary continuous speech recognition," in *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, 2011.

[93] J. Pan, C. Liu, Z. Wang, Y. Hu, and H. Jiang, "Investigation of deep neural networks (DNN) for large vocabulary continuous speech recognition: Why DNN surpasses GMMs in acoustic modeling," in *Proc. ISCSLP*, 2012.

[94] B. Kingsbury, "Lattice-based optimization of sequence classification criteria for neural-network acoustic modeling," in *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*, April 2009, pp. 3761–3764.

[95] K. Veselỳ, A. Ghoshal, L. Burget, and D. Povey, "Sequence-discriminative training of deep neural networks." in *INTERSPEECH*, 2013, pp. 2345–2349.

[96] Y. Tachioka and S. Watanabe, "Discriminative training of acoustic models for system combination." in *INTERSPEECH*, 2013, pp. 2355–2359.

[97] B. Kingsbury, T. N. Sainath, and H. Soltau, "Scalable minimum bayes risk training of deep neural network acoustic models using distributed hessian-free optimization." in *INTERSPEECH*, 2012.

[98] H. Su, G. Li, D. Yu, and F. Seide, "Error back propagation for sequence training of context-dependent deep networks for conversational speech transcription." in *ICASSP*, 2013, pp. 6664–6668.

[99] Y. Zhang, E. Chuangsuwanich, and J. Glass, "Extracting deep neural network bottleneck features using low-rank matrix factorization," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, May 2014, pp. 185–189.

[100] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time-series," in *The Handbook of Brain Theory and Neural Networks*, M. A. Arbib, Ed. MIT Press, 1995.

[101] Y. LeCun, F. Huang, and L. Bottou, "Learning methods for generic object recognition with invariance to pose and lighting," in *Proceedings of CVPR'04*, 2004.

[102] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, 2012, pp. 1106–1114.

[103] H. Lee, P. Pham, Y. Largman, and A. Ng, "Unsupervised feature learning for audio classification using convolutional deep belief networks," in *Advances in Neural Information Processing Systems 22*, 2009, pp. 1096–1104.

[104] D. Hau and K. Chen, "Exploring hierarchical speech representations using a deep convolutional neural network," in *the 11th UK Workshop on Computational Intelligence (UKCI 2011)*, Manchester, UK, 2011.

[105] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, and G. Penn, "Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*. IEEE, 2012, pp. 4277–4280.

[106] T. N. Sainath, A.-R. Mohamed, B. Kingsbury, and B. Ramabhadran, "Deep convolutional neural networks for LVCSR," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2013.

[107] L. Deng, O. Abdel-Hamid, and D. Yu, "A deep convolutional neural network using heterogeneous pooling for trading acoustic invariance with phonetic confusion," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2013.

[108] O. Abdel-Hamid, L. Deng, and D. Yu, "Exploring convolutional neural network structures and optimization techniques for speech recognition," in *INTERSPEECH*, 2013.

[109] O. Abdel-Hamid, A.-R. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *Audio, Speech, and Language Processing, IEEE/ACM Transactions on*, vol. 22, no. 10, pp. 1533–1545, Oct 2014.

[110] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *Proceedings of the 20th international conference on Artificial neural networks:*

Part III, ser. ICANN'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 92–101.

[111] K. F. Lee and H. W. Hon, "Speaker-independent phone recognition using hidden Markov models," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 37, no. 11, pp. 1641 – 1648, November 1989.

[112] S. J. Young, G. Evermann, M. J. F. Gales, T. Hain, D. Kershaw, G. Moore, J. Odell, D. Ollason, D. Povey, V. Valtchev, and P. C. Woodland, *The HTK Book, version 3.4*. Cambridge, UK: Cambridge University Engineering Department, 2006.

[113] "Hidden Markov model toolkit (HTK)," http://htk.eng.cam.ac.uk.

[114] L. Gillick and S. J. Cox, "Some statistical issues in the comparison of speech recognition algorithms," in *Acoustics, Speech, and Signal Processing, 1989. ICASSP-89., 1989 International Conference on*. IEEE, 1989, pp. 532–535.

[115] J.-L. Gauvain and C.-H. Lee, "Maximum a posteriori estimation for multivariate gaussian mixture observations of markov chains," *Speech and Audio Processing, IEEE Transactions on*, vol. 2, no. 2, pp. 291 –298, apr 1994.

[116] S. M. Ahadi and P. C. Woodland, "Combined bayesian and predictive techniques for rapid speaker adaptation of continuous density hidden markov models," *Computer Speech and Language*, vol. 11, no. 3, pp. 187 – 206, 1997.

[117] C. J. Leggetter and P. C. Woodland, "Maximum likelihood linear regression for speaker adaptation of continuous density hidden Markov models," *Computer Speech and Language*, vol. 9, no. 2, pp. 171 – 185, 1995.

[118] M. Gales, "Maximum likelihood linear transformations for HMM-based speech recognition," *Computer Speech and Language*, vol. 12, no. 2, pp. 75 – 98, 1998.

[119] V. Digalakis, D. Rtischev, and L. Neumeyer, "Speaker adaptation using constrained estimation of gaussian mixtures," *Speech and Audio*

Processing, IEEE Transactions on, vol. 3, no. 5, pp. 357 –366, sep 1995.

[120] P. C. Woodland, "Speaker adaptation for continuous density HMMs: A review," in *ISCA ITR-Workshop on Adaptation Methods for Speech Recognition*, Aug. 2001, pp. 11–19.

[121] L. Lee and R. Rose, "Speaker normalization using efficient frequency warping procedures," in *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on*, vol. 1, may 1996, pp. 353 –356 vol. 1.

[122] G. Saon, M. Padmanabhan, R. Gopinath, and S. Chen, "Maximum likelihood discriminant feature spaces," in *Acoustics, Speech, and Signal Processing, 2000. ICASSP'00. Proceedings. 2000 IEEE International Conference on*, vol. 2. IEEE, 2000, pp. II1129–II1132.

[123] J. Neto, L. Almeida, M. Hochberg, C. Martins, L. Nunes, S. Renals, and T. Robinson, "Speaker-adaptation for hybrid hmm-ann continuous speech recognition system," in *Eurospeech*, Madrid, Spain, 9 1995, pp. 2171–2174.

[124] V. Abrash, H. Franco, A. Sankar, and M. Cohen, "Connectionist speaker normalization and adaptation," in *Eurospeech*, 1995, pp. 2183–2186.

[125] J. S. Bridle and S. Cox, "Recnorm: Simultaneous normalisation and classification applied to speech recognition," in *Advances in Neural Information Processing Systems 3*, R. Lippmann, J. Moody, and D. Touretzky, Eds. Morgan-Kaufmann, 1991, pp. 234–240.

[126] R. Gemello, F. Mana, S. Scanzio, P. Laface, and R. D. Mori, "Linear hidden transformations for adaptation of hybrid ANN/HMM models," *Speech Communication*, vol. 49, no. 10-11, pp. 827 – 835, 2007.

[127] B. Li and K. C. Sim, "Comparison of discriminative input and output transformations for speaker adaptation in the hybrid NN/HMM systems." in *INTERSPEECH*, 2010, pp. 526–529.

[128] S. M. Siniscalchi, J. Li, and C.-H. Lee, "Hermitian based hidden activation functions for adaptation of hybrid HMM/ANN models," in *INTERSPEECH*, 2012.

[129] K. Yao, D. Yu, F. Seide, H. Su, L. Deng, and Y. Gong, "Adaptation of context-dependent deep neural networks for automatic speech recognition." in *SLT*, 2012, pp. 366–369.

[130] D. Yu, K. Yao, H. Su, G. Li, and F. Seide, "Kl-divergence regularized deep neural network adaptation for improved large vocabulary speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013, pp. 7893–7897.

[131] S. Scanzio, P. Laface, R. Gemello, and F. Mana, "Adaptation of hybrid ANN/HMM using weights interpolation," in *ICASSP 2006*, vol. 5, May 2006.

[132] H. Liao, "Speaker adaptation of context dependent deep neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, May 2013, pp. 7947–7951.

[133] S. Dupont and L. Cheboub, "Fast speaker adaptation of artificial neural networks for automatic speech recognition," in *ICASSP 2000*, vol. 3, 2000, pp. 1795–1798.

[134] O. Abdel-Hamid and H. Jiang, "Fast speaker adaptation of hybrid NN/HMM model for speech recognition based on discriminative learning of speaker code," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, May 2013.

[135] S. Xue, O. Abdel-Hamid, H. Jiang, and L. Dai, "Direct adaptation of hybrid DNN/HMM model for fast speaker adaptation in LVCSR based on speaker code," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, May 2014, pp. 6339–6343.

[136] S. Xue, O. Abdel-Hamid, H. Jiang, L. Dai, and Q. Liu, "Fast adaptation of deep neural network based on discriminant codes for speech recognition," *Audio, Speech, and Language Processing, IEEE/ACM Transactions on*, vol. 22, no. 12, pp. 1713–1725, Dec 2014.

[137] O. Abdel-Hamid and H. Jiang, "Rapid and effective speaker adaptation of convolutional neural network based models for speech recognition," in *INTERSPEECH*, 2013.

[138] N. Strom, "Speaker adaptation by modeling the speaker variation in a continuous speech recognition system," in *ICSLP 96*, vol. 2, Oct 1996, pp. 989–992.

[139] D. Yu, X. Chen, and L. Deng, "Factorized deep neural networks for adaptive speech recognition," in *Int. Workshop on Statistical Machine Learning for Speech Processing*, 2012.

[140] G. Saon, H. Soltau, D. Nahamoo, and M. Picheny, "Speaker adaptation of neural network acoustic models using i-vectors," in *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, Dec 2013, pp. 55–59.

[141] V. Gupta, P. Kenny, P. Ouellet, and T. Stafylakis, "I-vector-based speaker adaptation of deep neural networks for french broadcast audio transcription," in *Acoustics, Speech and Signal Processing, 2014. ICASSP 2014. IEEE International Conference on*, 2014.

[142] M. Ostendorf, V. Digalakis, and O. Kimball, "From HMM's to segment models: A unified view of stochastic modeling for speech recognition," *Speech and Audio Processing, IEEE Transactions on*, vol. 4, no. 5, pp. 360–378, 1996.

[143] L. Deng, M. Aksmanovic, X. Sun, and C. Wu, "Speech recognition using hidden Markov models with polynomial regression functions as nonstationary states," *Speech and Audio Processing, IEEE Transactions on*, vol. 2, no. 4, pp. 507 –520, oct 1994.

[144] H. Gish and K. Ng, "A segmental speech model with applications to word spotting," in *Acoustics, Speech, and Signal Processing, 1993. ICASSP-93., 1993 IEEE International Conference on*, vol. 2, April 1993, pp. 447–450.

[145] S. Roucos, M. Ostendorf, H. Gish, and A. Derr, "Stochastic segment modelling using the estimate-maximize algorithm," in *International Conference on Acoustics, Speech, and Signal Processing, 1988. ICASSP-88.*, vol. 1, Apr 1988, pp. 127–130.

[146] M. Ostendorf, I. Bechwati, and O. Kimball, "Context modeling with the stochastic segment model," in *IEEE International Conference on Acoustics, Speech, and Signal Processing, 1992. ICASSP-92.*, vol. 1, Mar 1992, pp. 389–392.

[147] O. Kimball and M. Ostendorf, "On the use of tied-mixture distributions," in *Proceedings of the Workshop on Human Language Technology*, 1993, pp. 102–107.

[148] C. Wellekens, "Explicit time correlation in hidden Markov models for speech recognition," in *IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP '87.*, vol. 12, Apr 1987, pp. 384–386.

[149] V. Digalakis, J. Rohlicek, and M. Ostendorf, "A dynamical system approach to continuous speech recognition," in *International Conference on Acoustics, Speech, and Signal Processing, 1991. ICASSP-91.*, vol. 1, Apr 1991, pp. 289–292.

[150] M. Saerens and H. Bourlard, "Linear and nonlinear prediction for speech recognition with hidden Markov models." in *EUROSPEECH*, 1993.

[151] E. Levin, "Word recognition using hidden control neural architecture," in *International Conference on Acoustics, Speech, and Signal Processing, 1990. ICASSP-90.* IEEE, 1990, pp. 433–436.

[152] J. R. Glass, "A probabilistic framework for segment-based speech recognition," *Computer Speech & Language*, vol. 17, no. 2-3, pp. 137–152, 2003.

[153] G. Zweig and P. Nguyen, "A segmental CRF approach to large vocabulary continuous speech recognition," in *Automatic Speech Recognition Understanding, 2009. ASRU 2009. IEEE Workshop on*, 13 2009-dec. 17 2009, pp. 152 –157.

[154] Y. He and E. Fosler-Lussier, "Efficient segmental conditional random fields for one-pass phone recognition," in *INTERSPEECH 2012*, 2012.

[155] G. Zweig, "Classification and recognition with direct segment models," in *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, march 2012, pp. 4161 –4164.

[156] O. Abdel-Hamid, L. Deng, D. Yu, and H. Jiang, "Deep segmental neural networks for speech recognition." in *INTERSPEECH*, 2014.

[157] S. Austin, G. Zavaliagkos, J. Makhoul, and R. Schwartz, "Continuous speech recognition using segmental neural nets," in *Neural Networks, 1992. IJCNN., International Joint Conference on*, vol. 2, jun 1992, pp. 314 –319 vol.2.

[158] L. Tóth, "Combining time- and frequency-domain convolution in convolutional neural network-based phone recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, May 2014, pp. 190–194.

Appendix A: List of Authored Publication During PhD Study

1. Refereed journal papers

- Shaofei Xue, **Ossama Abdel-Hamid**, Hui Jiang, Lirong Dai, Qingfeng Liu, "Fast Adaptation of Deep Neural Network Based on Discriminant Codes for Speech Recognition," *Audio, Speech, and Language Processing, IEEE/ACM Transactions on* , vol.22, no.12, pp.1713-1725, Dec. 2014

- **Ossama Abdel-Hamid**, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu, "Convolutional Neural Networks for Speech Recognition," *Audio, Speech, and Language Processing, IEEE/ACM Transactions on* , vol.22, no.10, pp.1533,1545, Oct. 2014

- Eric P. Wilkinson, **Ossama Abdel-Hamid**, John J. Galvin, Hui Jiang, and Fu Qian-Jie, "Voice conversion in cochlear implantation," *The Laryngoscope*, 2013.

2. Refereed conference papers

- Shaofei Xue, **Ossama Abdel-Hamid**, Hui Jiang, and Li-Rong Dai, "Direct Adaptation of Hybrid DNN/HMM Model for Fast Speaker Adaptation in LVCSR Based on Speaker Code," in *ICASSP*, 2014

- **Ossama Abdel-Hamid** and Hui Jiang, "Rapid and Effective Speaker Adaptation of Convolutional Neural Network Based Models for Speech Recognition," in *INTERSPEECH*, 2013

- **Ossama Abdel-Hamid**, Li Deng, Dong Yu, "Exploring Convolutional Neural Network Structures and Optimization Techniques for Speech Recognition," in *INTERSPEECH*, 2013

- **Ossama Abdel-Hamid**, Li Deng, Dong Yu, Hui Jiang, "Deep Segmental Neural Networks for Speech Recognition," in *INTERSPEECH*, 2013

- **Ossama Abdel-Hamid** and Hui Jiang, "Fast speaker adaptation of hybrid NN/HMM model for speech recognition based on discriminative learning of speaker code," in *ICASSP*, 2013

- Li Deng, **Ossama Abdel-Hamid**, Dong Yu, "A deep convolutional neural network using heterogeneous pooling for trading acoustic invariance with phonetic confusion," in *ICASSP* 2013

- **Ossama Abdel-Hamid**, Abdel-rahman Mohamed, Hui Jiang, Gerald Penn, "Applying Convolutional Neural Networks concepts to Hybrid NN-HMM Model for speech recognition," in *ICASSP*, 2012.