Exploring Deep Learning Methods for discovering features in speech signals.

by

Navdeep Jaitly

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Abstract

Exploring Deep Learning Methods for discovering features in speech signals.

Navdeep Jaitly
Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto
2014

This thesis makes three main contributions to the area of speech recognition with Deep Neural Network - Hidden Markov Models (DNN-HMMs).

Firstly, we explore the effectiveness of features learnt from speech databases using Deep Learning for speech recognition. This contrasts with prior works that have largely confined themselves to using traditional features such as Mel Cepstral Coefficients and Mel log filter banks for speech recognition. We start by showing that features learnt on raw signals using Gaussian-ReLU Restricted Boltzmann Machines can achieve accuracy close to that achieved with the best traditional features. These features are, however, learnt using a generative model that ignores domain knowledge. We develop methods to discover features that are endowed with meaningful semantics that are relevant to the domain using capsules. To this end, we extend previous work on transforming autoencoders and propose a new autoencoder with a domain-specific decoder to learn capsules from speech databases. We show that capsule instantiation parameters can be combined with Mel log filter banks to produce improvements in phone recognition on TIMIT. On WSJ the word error rate does not improve, even though we get strong gains in classification accuracy. We speculate this may be because of the mismatched objectives of word error rate over an utterance and frame error rate on the sub-phonetic class for a frame.

Secondly, we develop a method for data augmentation in speech datasets. Such methods result in strong gains in object recognition, but have largely been ignored in speech recognition. Our data augmentation encourages the learning of invariance to vocal tract length of speakers. The method is shown to improve the phone error rate on TIMIT and the word error rate on a 14 hour subset of WSJ.

Lastly, we develop a method for learning and using a longer range model of targets, conditioned on the input. This method predicts the labels for multiple frames together and uses a geometric average of these predictions during decoding. It produces state of the art results on phone recognition with TIMIT and also produces significant gains on WSJ.

Acknowledgements

I want to thank Ryan Lilien for encouraging me to come to University of Toronto. Were it not for him, I would not have been blessed with one of the rarest of gifts in life — a teacher and mentor with an unmatched combination of intellect, intuition, wit and kindness. From my teacher and mentor, Geoffrey Hinton, I hope I have learnt not only how the brain works, but how a phenomenal scientist can also be a phenomenal human being.

I want to thank all of the members of the Machine Learning and affiliated groups at UofT that have made the last few years so enjoyable — Charlie, Danny, Laurent, Fernando, Jasper, Kevin, Yujia, Ryan (Kiros and Adams), Jake, Hugo, Marc'Aurelio, Vlad, Andriy, Nitish, Rus, Amit, Tanya, Vinod, Graham, Maks, James, Tijmen, Marcus, Jonathan, Ilya, George, Tong, Olya, Alex (Graves, Krizhevsky and Schwing), Mohammed, Chris, Jimmy, Abdel-Rahman and others. Special thanks to those of you who made trips to 'Thirsty and Miserable' so much fun (you know who you are).

Thank you, also, to the Comp Bio group members with whom I started this journey: Marc, Marya, Abe, Izhar, Misko (Michael), Nilgun, Mike Brudno, Ryan Lilien and Vlad.

I also want to thank my current and former committee members - Frank Rudzicz, Gerald Penn and Radford Neal for being generous with their time and so prompt with their feedback.

Extra thanks to my office-mates - Volodymyr Mnih (Vlad) for being such a great friend, George Dahl for your ever-ready wit and free, but insightful, advice in all matters and Nitish Srivastava for your willingness to entertain new ideas. It has been wonderful sharing offices, time and ideas with all of you.

Special thanks to Relu Patrascu not just for your excellent management of the computing infrastructure, but also your friendship — sorry I don't like Bollywood movies as much as you do :).

Thank you, Ilya Sutskever, for your infectious enthusiasm and eternal optimism. Holy s!*t, no one quite admires cool receptive filters like you do! Alex Krizhevsky, thank you for winning ImageNET and forming DNNResearch — I will be forever grateful.

A special thanks to Dan Povey — KALDI has been a real gift to the community and will continue to enable our research directions for years to come. Also, thanks to other members of the speech community outside Toronto whose work has influenced and inspired me - Karel Vesely, Tara Sainath, Brian Kingsbury, Li Deng, Dong Yu, Vincent Vanhoucke, Oriol Vinyals and Andrew Senior, to name a few.

Thanks also to my manager during my internship at Google — Vincent Vanhoucke. Your brand of pragmatism, brilliance and efficiency is hard to come by.

Rich Zemel, thank you for the weekly squash games — I could not have wished for a better squash partner. You have not only been a teacher but a friend. I will miss our Monday morning games. Also, thank you for the Z-group meetings — I wish I had not waited four years before joining.

Lastly, and most importantly, I want to thank my family; you have selflessly shared me with my work over evenings and weekends through various deadlines. Thank you, Bhairavi, my best friend and spouse; without you this journey would neither have started nor have ended. My lovely son, Unkush, your RNN language models are both amusing and astounding — clearly they are not coded on your Y chromosome. And my intense daughter, Sunkhya, may your learning rate and momentum remain high.

Contents

I Background **1**

1 Outline **2**

2 Speech Recognition With a DNN-HMM System **5**
- 2.1 An Overview of Speech Recognition . . . 5
- 2.2 Frame-based Features for Speech Recognition . . . 6
- 2.3 GMM-HMM Speech Recognizers . . . 7
  - 2.3.1 Gaussian Mixture Models . . . 7
  - 2.3.2 Hidden Markov Models . . . 8
  - 2.3.3 GMM-HMMs for Speech Recognition . . . 9
- 2.4 DNN-HMM Speech Recognizers . . . 10
- 2.5 Baselines . . . 12
  - 2.5.1 TIMIT . . . 12
  - 2.5.2 Wall Street Journal . . . 12

II Unsupervised Feature Learning **15**

3 Learning Features From a Generative Model of Raw Speech Signals **16**
- 3.1 Probabilistic Model . . . 17
- 3.2 Learning . . . 18
  - 3.2.1 RBM Training . . . 19
  - 3.2.2 RBM Results . . . 19
- 3.3 Speech Recognition Experiments . . . 24
  - 3.3.1 Dimensionality Reduction . . . 28
- 3.4 Conclusion . . . 28

4 Learning Features Using Transforming Autoencoders **29**
- 4.1 Transforming Autoencoders . . . 30
- 4.2 Application to Speech Signals . . . 31
  - 4.2.1 Waveforms . . . 31
  - 4.2.2 Spectrograms . . . 31
- 4.3 A Way to Improve Transforming Autoencoders . . . 32
- 4.4 Results and Discussion . . . 32
  - 4.4.1 Autoencoder on Raw Signals . . . 32

4.4.2 Autoencoder on Spectrograms . . . 36
4.4.3 Impact of Regularization . . . 39
4.4.4 Peering Into the Mind of the Capsules . . . 39
4.4.5 Equivariance of Fantasy . . . 41
4.4.6 Impact of Number of Units . . . 41
4.4.7 Application to Phone Recognition . . . 41
4.5 Future Directions to Explore . . . 44

5 Learning Features Using Deformable Templates. 45
5.1 Autoencoder With Templated Parts . . . 45
5.2 Application to Frames of Spectrograms . . . 48
5.2.1 Motivation . . . 48
5.2.2 Autoencoder Details . . . 49
5.2.3 Experimental Methods and Results . . . 51
5.3 Application to FBANKS Patches . . . 56
5.3.1 Autoencoder Details . . . 56
5.3.2 Results . . . 57
5.4 Significance Testing of Results . . . 61
5.5 Discussion . . . 61
5.5.1 Discord Between Training Objective and Speech Recognition Accuracy . . . 62
5.5.2 More Powerful Decoders . . . 62
5.5.3 Better Use of Instantiation Parameters . . . 63

III Data Augmentation for Learning Invariances 64

6 Vocal Tract Length Perturbation 65
6.1 Introduction . . . 65
6.2 Methods . . . 66
6.2.1 Mel Filter Banks . . . 66
6.2.2 VTLN . . . 69
6.2.3 VTLP . . . 69
6.2.4 Neural Network Training . . . 70
6.2.5 Decoding . . . 70
6.3 Experiments . . . 71
6.3.1 Fully Connected DNNs . . . 71
6.3.2 Convolutional Neural Network (CNN) . . . 72
6.4 Results . . . 72
6.4.1 Significance Testing of Results . . . 76
6.5 Discussion . . . 76
6.6 Conclusions . . . 76

IV Partial Structure Learning **77**

7 Using Long Range Predictions **78**
7.1 Introduction . . . 79
7.2 Methods . . . 81
7.2.1 Multi-frame Target Training . . . 81
7.2.2 Decoding With Autoregressive Targets (DART) . . . 82
7.3 Experiments With FBANK Inputs . . . 82
7.3.1 Effect of Context Window Sizes . . . 83
7.3.2 Frame Error Rate of Models . . . 84
7.3.3 Impact of Depth of DNNs . . . 84
7.3.4 WSJ Results . . . 85
7.3.5 Geometric Averaging Compared to Arithmetic Averaging . . . 86
7.4 Experiments With Methods Developed in This thesis . . . 86
7.4.1 TIMIT . . . 86
7.4.2 WSJ . . . 87
7.5 Conclusions . . . 87

8 Conclusions **89**
8.1 Summary of Contributions . . . 89
8.2 Analysis of Errors . . . 90
8.3 Future Avenues for Exploration . . . 91

A Breakdown of Phone Error Rates **93**

Bibliography **93**

Part I

Background

Chapter 1

Outline

Progress in Deep Learning has recently revitalized the use of neural networks for speech recognition [Hinton, Deng, Yu, Dahl, Mohamed, Jaitly, Senior, Vanhoucke, Nguyen, Sainath, and Kingsbury, 2012a]. The state of the art in the area has been improved significantly from adoption of these methods in training Deep Neural Network - Hidden Markov Model (DNN-HMM) systems [Mohamed et al., 2012, Dahl et al., 2010, Jaitly et al., 2012, Sainath et al., 2013, Deng et al., 2013]. The use of these methods has however been largely confined to using more layers in neural networks and the use of convolutional neural networks on features that were hand engineered by the speech community. Surprisingly, some of the other core techniques used in Deep Learning have not been applied to speech. This thesis applies three main ideas from Deep Learning to speech. Firstly, we use Deep Learning methods for unsupervised discovery of features in speech signals [Jaitly and Hinton, 2011a,b, 2013a]. These features are then used for speech recognition with a DNN-HMM system. Secondly, we develop a data augmentation strategy to improve recognition accuracy [Jaitly and Hinton, 2013b]. Lastly, we model longer range structure than is done traditionally using multi-frame predictions to improve speech recognition accuracy [Jaitly and Hinton, 2014]. We briefly motivate each of these areas before describing the individual chapters.

The speech community has hand engineered speech features such as Mel Log filter banks (FBANKs), Mel frequency Cepstral Coefficients (MFCCs) [Davis and Mermelstein, 1980], Perceptual Linear Prediction (PLPs) [Hermansky, 1990], etc, over the last 40 years based on acoustic, psychophysical and anatomical factors. These features have two main characteristics that make them suitable for use in speech recognition systems - they are a low-dimensional representation of speech and they are very effective at preserving the information required to achieve high speech recognition accuracy. As a result, such 'front-end' preprocessing of speech signals is an essential component of every speech recognition system.

With the increase in computational power and the availability of large speech corpora, it should be possible to learn features from speech databases automatically. In fact, a significant amount of research in Deep Learning has gone into developing unsupervised models of images. Work on speech recognition however has been confined to using Mel log filter banks as inputs to neural networks, with or without pretraining with a Deep Belief Network (DBN) [Hinton et al., 2006]. On object detection tasks it was seen that working from raw pixels directly provided the best results. The first part of this thesis is an exploration of methods to learn alternate representations of audio data for use with DNN-HMM systems. The effectiveness of these features for speech recognition is tested by using them as the inputs

to a DNN-HMM recognizer.

Data augmentation strategies play a crucial role in object detection techniques using Deep Learning because they encourage models to learn important invariances. However, these methods have, surprisingly, not been exploited much in speech recognition. The second part of this thesis presents a way to learn invariance to vocal tract lengths of speakers using a data augmentation strategy.

DNN-HMM systems typically make predictions of sub-phonetic classes for single frames. As such they ignore long range structure. Sequence discriminative methods avoid this trap by focussing on the long range objective of achieving a correct transcription from the speech recognizer. However these methods are cumbersome because they require decoding speech during training, which is computationally expensive. In the last part of this thesis we explore a method that makes predictions for multiple frames during training, and as a result attempts to build a longer range sequential model of targets conditioned on acoustics. During decoding we use these predictions convolutionally. This strategy significantly improves recognition accuracy regardless of the type of features that are used.

A brief outline of this work is as follows:

- **Speech recognition with a DNN-HMM system:** In chapter 2, we summarize how to train and use a DNN-HMM hybrid speech recognizer. We then set up the DNN-HMM speech recognition baselines for the thesis, using two different datasets - TIMIT [Garofolo et al., 1993] and Wall Street Journal [Paul and Baker, 1992]. Both these baselines use speech features — FBANKs and MFCCs — in the front-end.

- **Learning features from raw speech signals:** In chapter 3 we explore whether features can be learnt directly from unprocessed waveforms [Jaitly and Hinton, 2011a]. For this purpose we use Gaussian-Stepped Sigmoid Unit Restricted Boltzmann Machines, trained with Contrastive Divergence to learn features. We use these features to perform speech recognition. We see that the features we learnt produce better speech recognition accuracy than MFCCs, but worse than Mel log filter banks. Our success in learning features directly from raw signals suggests that features may indeed be learnt from speech data directly. The features in this chapter however ignore domain knowledge both because the frequency characteristics of signals are difficult to ascertain directly from raw signals — especially for turbulent flow — and because of the way the RBMs were used.

- **Learning features using transforming autoencoders** In chapter 4 we use Transforming Autoencoders [Hinton et al., 2011] to discover features in spectrograms and raw speech signals that are described by a rich set of descriptors called *capsules* that are relevant to the domain of speech [Jaitly and Hinton, 2011b]. Capsules were seen to capture interesting features in the data, but did not produce appreciable improvements in recognition accuracy over Mel log filter banks. We hypothesize that this is partly because of cooperation between the encoder and decoder parts of the transforming autoencoder.

- **Learning features with deformable templates:** In chapter 5 we reduce cooperation between the encoding and decoding parts of Transforming Autoencoders by replacing the generic neural network based decoder with a function that deforms and adds together a set of templates to create a reconstruction [Jaitly and Hinton, 2013a]. We applied this method to modelling frames of spectrograms and patches of Mel log filter banks. The features from this model lead to improved accuracy in speech recognition, when combined with filter bank features. Features learnt on Mel

log filter bank patches generally outperform the features learnt on frames of spectrograms. This is probably due to the inductive bias produced by using features that mimic the frequency characteristics of the auditory apparatus. It is our assessment that further feature learning techniques should either build on Mel log filter bank patches, or at least attempt to incorporate the variable frequency resolution of the Mel scale, or other scales such as the Barq scale which mimic the experimentally observed resolution of the human auditory system.

- **Vocal Tract Length Perturbation:** In chapter 6 we develop a method we call Vocal Tract Length Perturbation (VTLP) to augment Mel Log filter bank training data [Jaitly and Hinton, 2013b]. VTLP uses Vocal Tract Length Normalization (VTLN) — a technique that is used to normalize differences between speakers that arise primarily due to differences in vocal tract length. However, it uses VTLN not to normalize data, but instead to generate random distortions, creating an augmented dataset. A neural network trained on the augmented data learns to be invariant to vocal tract length differences. VTLP produces consistent improvements in accuracy on small datasets using DNNs of different depths.

- **Sequence Level Training:** The above chapters use the traditional method of training a DNN-HMM system - the DNN is trained separately from the dynamic HMM model. As a result, there is a slight disconnect in the objective of the overall system and the objective of the neural network training — the former tries to reduce word error rate while the latter tries to reduce frame-by-frame cross entropy. As such, it leaves open the question of how our features compare to traditional features when a better objective function is optimized. We use this motivation to develop a novel training procedure for the hybrid model that leads to significant improvements in accuracy of the speech recognizers [Jaitly and Hinton, 2014]. We use this method to retrain the hybrid systems above.

Chapter 2

Speech Recognition With a DNN-HMM System

In this chapter we describe the baselines we use for this thesis - Deep Neural Network - Hidden Markov Model (DNN-HMM) hybrid speech recognizers that use MFCC and FBANK features as inputs. In the subsequent chapters, we will compare the accuracy of these systems with that of DNN-HMM recognizers built using features we learn with Deep Learning methods.

DNN-HMM recognizers are built on top of Gaussian Mixture Model - Hidden Markov Model (GMM-HMM) recognizers, so we first discuss a GMM-HMM recognizer. Then we explain how Deep Neural Network models can be used to improve the acoustic models of GMM-HMM models. The presentation of these ideas here is only as detailed as is required to understand the rest of the thesis; more detailed surveys of GMM-HMMs and DNN-HMMs can be found elsewhere ([Young, 2001, Young et al., 2002] and [Morgan and Bourlard, 1995, Trentin and Gori, 2001] respectively).

2.1 An Overview of Speech Recognition

Figure 2.1 shows the components of a general speech recognition system, which we describe here briefly.

An input utterance (waveform) is first preprocessed using an *audio front-end* that extracts acoustic features. Most speech recognizers (including GMM-HMMs, DNN-HMMs that we consider in this thesis) use a frame-based model in which an input waveform is converted into a sequence of frames of features of equal duration (see section 2.2 for more details). Features such as FBANKs, MFCCs [Davis and

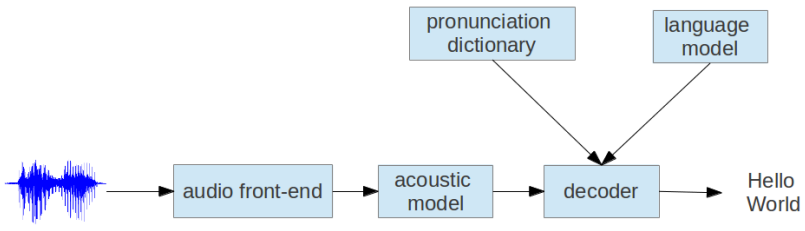


Figure 2.1: An overview of speech recognition

Mermelstein, 1980], Linear Predictive Coding (LPCs) [O'Shaughnessy, 1988], PLPs [Hermansky, 1990] are examples of this category of features. However, speech recognizers have also been developed that use features computed over varying length segments [Glass, 2003]. Our current body of work deals only with frame-based features, although some of the ideas, such as transforming autoencoders and deformable templates (chapters 4 and 5) could also be applied to segment-based approaches.

The *acoustic model* is a statistical model of the features computed by the acoustic front-end. Typically, the statistical model is a generative model (such as the GMM considered below) of the features conditioned on different spoken sound classes or *phonemes*. The model is used to compute the likelihood of generating an acoustic observation from a hypothesized transcript.

The *pronunciation dictionary* maps each word in the chosen language to its pronunciation - a sequence of phonemes. Frequently a word may have multiple alternative pronunciations, in which case the pronunciation dictionary has multiple entries for a word. The dictionary is used to provide constraints on sequences of sounds that are possible, and is an integral part of speech recognizers. Without the constraints provided by a dictionary, it is unlikely that recognizers would be successful at producing a reasonable decoding.

The *language model* is a statistical model of word sequences in the chosen language and is used to provide a relative ranking of different word sequences. Most speech recognizers use N-gram language models because they are easy to incorporate with other components - their low order Markov assumption makes it easy to use them in beam search routines [Jurafsky et al., 2000, Mohri et al., 2002]. However, advanced language models such as Recurrent Neural Language Models can be used in cascades with N-gram language models to refine transcripts produced by speech recognizers in a first pass [Mikolov et al., 2011].

The *decoder* is the module of the speech recognizer that transcribes an input utterance. It does this by generating candidate transcriptions that respect the constraints of the pronunciation dictionary and the language model. Each candidate is scored using scores from the acoustic model, representing the likelihood of the observation given the candidate, and scores from the language model, representing a prior probability of the candidate (i.e. the word sequence that the transcript represents). We must note that the process of generating candidate transcriptions is necessarily tightly coupled with the scoring of inputs by the acoustic model and the word sequences by the language model, for real world tasks. Generating and scoring all candidates one by one is not feasible computationally because of a combinatorial increase in number of candidates - most decoders use a combination of dynamic programming and beam-search to generate a subset of plausible candidates and score them at same time [Soong and Huang, 1991, Ney and Ortmanns, 1999, Mohri et al., 2002].

2.2 Frame-based Features for Speech Recognition

As mentioned in section 2.1, we use frame-based features in this thesis. The computation of the various types of commonly used frame-based features is inherently similar and we describe this here.

Speech signals are recordings of air pressure arriving at a microphone, that are digitized at a fixed rate. Feature representations of speech need to be able to capture both the dynamics of speech, which change at a rate determined by articulatory organs such as tongue, glottis, jaw etc, and the instantaneous frequency characteristics. To do this, the sequence is broken up into frames, that start at regular stride

intervals, S, and have the same window size W[1]. Thus, given a real valued speech signal $x_0, x_2, \cdots x_{T-1}$, frame i is the subsequence $y_{0\cdots(W-1)} = x_{iS\cdots(iS+W-1)}$. Each frame, i, is used to compute instantaneous characteristics of speech and the sequence of frames is used to capture dynamic properties of speech. Feature types differ in the computations they perform on these windows. For example, LPC features are computed by finding coefficients of an autoregressive model of order D on the samples of the frame. i.e. $y_t = \sum_{k=1}^{D} c_k y_{t-k}$. The coefficients $c_1 \cdots c_D$ are used as the dimensions of the feature vector, $\mathbf{v}_i = [c_1 c_2 \cdots c_D]^T$, for the i^{th} frame. For FBANK features, a Discrete Fourier Transform (DFT) is first computed for a frame. A weighted sum of the energy in different frequency bands is computed and log transformed. Each different bin in the data vector $\mathbf{v}_i$ corresponds to a different frequency band, with triangular weights being applied over the band [Davis and Mermelstein, 1980].

For speech applications a stride and window of 10-30 ms produce best results in speech recognition tasks [Picone, 1993, Rabiner and Schafer, 1979]. At these ranges, a reasonable compromise can be achieved between between capturing dynamics of speech and estimating features accurately. Most speech recognition systems use 25 ms frame windows and a stride of 10 ms [Povey et al., 2011b, Soltau et al., 2010, Schalkwyk et al., 2010, Walker et al., 2004, Young et al., 2002]. At a sampling rate of 16 kHz this represents 400 and 160 samples respectively. For this thesis will shall be using this window size and stride.

2.3 GMM-HMM Speech Recognizers

Gaussian Mixture Model - Hidden Markov Models (GMM-HMM) form the core of most speech recognizers. We describe GMMs, HMMs and GMM-HMMs here to the extent necessary to understand DNN-HMM hybrid models. For an in-depth treatment of these areas such as training of the model and application to speech recognition see [Bishop and Nasrabadi, 2006, Rabiner, 1989].

2.3.1 Gaussian Mixture Models

The Gaussian Mixture Model (GMM) is a model that generates real-valued vectors from one of M distinct Gaussian distributions. Formally, let $\mathbf{h}$ be a discrete latent variable which can take on one of M values with probabilities $\pi_1, \pi_2, \cdots, \pi_M$ respectively, where, $\sum_{m=1}^{M} \pi_m = 1$. Here, $\mathbf{h}$ can be represented as a vector, $(h_1, h_2, \cdots, h_M)$ where one and only one component is 1, and the rest are 0, i.e. $\sum_{m=1}^{M} h_m = 1, h_m \in \{0, 1\}$. Thus, the probability for a latent variable is, $p(\mathbf{h}) = \prod_{m=1}^{m=M} \pi_m^{h_m}$. Corresponding to each distinct value of m is a Gaussian probability distribution $p_m(\mathbf{v}; \boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)$ that defines the probability of generating an observation $\mathbf{v}$ from that component. Data is generated from this model by first generating a sample $\mathbf{h}$ for the latent variable, and then generating an observation from a Gaussian probability distribution, $p_m(\cdot; \boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)$ that corresponds to the category, m. The probability for an observation $\mathbf{v}$ can be calculated by marginalizing over the latent variables, i.e.

$$p(\mathbf{v}) = \sum_m \pi_m p_m(\mathbf{v}; \boldsymbol{\theta}_m) \tag{2.1}$$

where $\boldsymbol{\theta}_m = (\boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)$, $m = 1 \cdots M$, are the parameters of the mixture model. The log likelihood for this function is typically a non-convex function of the parameters and thus model-fitting by finding

[1]We note here that we will be using the term windows in the future, when we describe inputs to the DNNs. In that context *window* will refer to the number of frames that are used as inputs.

the global optimum of the likelihood function is not possible. However, the Expectation Maximization Algorithm [Baum et al., 1970, Dempster et al., 1977] is guaranteed to converge to a local optimum, by starting at some initial parameter setting and maximizing a lower bound on the log likelihood.

2.3.2 Hidden Markov Models

A Hidden Markov model is a discrete latent variable model used to model time sequences. At each time step, t, a latent variable $\mathbf{h}_t$, takes on one of K distinct states ($= i \in \{1, \cdots, K\}$) and emits an observation $\mathbf{v}_t$ for that time point from a conditional distribution, $p_i(\cdot; \boldsymbol{\theta}_i)$, called the emission distribution (which in the case of GMM-HMMs would be a GMM, as described in section 2.3.3). The latent variable $\mathbf{h}_t$ at any time follows a Markov property of order p - given the value of the latent variables $\mathbf{h}_{(t-p)}, \cdots \mathbf{h}_{(t-1)}$, p steps prior to t, the variable $\mathbf{h}_t$ is independent of the states at other prior times, i.e. $p\left(\mathbf{h}_t|\mathbf{h}_{1\cdots(t-1)}\right) = p\left(\mathbf{h}_t|\mathbf{h}_{(t-p)\cdots(t-1)}\right)$. The conditional distribution $p\left(\mathbf{h}_t|\mathbf{h}_{(t-p)\cdots(t-1)}\right)$ is called the transition probability. An HMM of order p can easily be converted to an HMM of order 1 by remapping combinations of states to individual states in a larger state space so we discuss only the case where $p = 1$. With $p = 1$ the transition probability can be represented as a matrix of parameters, $\mathbf{A}_{K\text{x}K}$, where $p(\mathbf{h}_t = i|\mathbf{h}_{t-1} = j) = A_{ji}$ is the probability that the hidden state at time t is i given that it was j at time $(t-1)$.

It can be seen that the marginal distribution at any time point is a mixture distribution of the conditional emission distributions. The prior probabilities of the emission distributions depend on the hidden state at the previous time point. Specifically, if the state at the previous time point was j, the conditional distribution at the current time point is $\sum_i A_{ji} p_i(\cdot; \boldsymbol{\theta}_i)$, and the marginal distribution is a mixture distribution of these conditional distributions weighted by the marginal probabilities of the states j at the previous time point. For the first time point, there is no prior state, so a full specification of the model thus requires one more component that is the vector of prior probabilities of the states at the first time point. We denote the prior probabilities by a vector $\boldsymbol{\pi} = (\pi_1, \cdots, \pi_K)$ on the probability simplex (i.e. $\sum_{k=1}^{K} \pi_k = 1$). Thus, $p(\mathbf{h}_1 = i) = \pi_i$.

Computing the probability of a given sequence $\mathbf{v}_1\mathbf{v}_2\cdots\mathbf{v}_T$ is more complicated for this model than it is for mixture models because the hidden states at each time step depend on the hidden states at other times steps. This has the effect of making any visible variables, $\mathbf{v}_t$, marginally dependent on the other visible variables $\mathbf{v}_{-t}$[2]. Thus, while one can easily calculate the marginal probability for any time point, $p(\mathbf{v}_t)$ using a mixture distribution with priors $(\mathbf{A}')^{t-1}\boldsymbol{\pi}$ and component distributions $p_k(\cdot; \boldsymbol{\theta}_k)$, $k = 1 \cdots K$, this does not help in computing the joint probability for a full sequence, as the joint probability is not the product of the individual marginal probabilities for the different times.

However, dynamic programming can be used to compute the likelihood by breaking up the computation into smaller tasks as follows. The joint probability for the subsequence starting at the first time step and having the hidden state, k, at the last time step, i.e. $p(\mathbf{v}_1 \cdots \mathbf{v}_t, \mathbf{h}_t = k)$ for $t = 2 \cdots T$ and $k = 1 \cdots K$, can be calculated using the recursion:

$$p(\mathbf{v}_1 \cdots \mathbf{v}_t, \mathbf{h}_t = k) = p_k(\mathbf{v}_t; \boldsymbol{\theta}_k) \sum_j p\left(\mathbf{v}_1 \cdots \mathbf{v}_{(t-1)}, \mathbf{h}_{t-1} = j\right) A_{jk} \tag{2.2}$$

with $p(\mathbf{v}_1, \mathbf{h}_1 = k) = p_k(\mathbf{v}_1; \boldsymbol{\theta}_k)\pi_k$. The probability of a sequence $\mathbf{v}_1 \cdots \mathbf{v}_T$ can be computed from this

[2]We use the notation $\mathbf{v}_{-t}$ to refer to latent variables at time points other than t

by marginalizing out the states at the last time point, i.e. $p(\mathbf{v}_1 \cdots \mathbf{v}_T) = \sum_{k=1}^{K} p(\mathbf{v}_1 \cdots \mathbf{v}_T, \mathbf{h}_T = k)$

Learning in these models is complicated by the fact that the log likelihood function is non-convex and it leads to many possible local maxima in parameter space. Optimization by simple gradient descent methods is slow because interactions between different parameters can lead to curvature in parameter space. The Expectation Maximization (EM) algorithm typically converges to a solution faster than simple gradient descent based procedures. However Salakhutdinov et al. [2003] showed that under certain conditions Conjugate Gradient (CG) converges faster than EM, and developed a hybrid algorithm that combines EM with CG.

A significant drawback of HMMs is that they can only model very simple patterns. Since the state of the hidden variable $\mathbf{h}_t$ is the only way to influence the distribution at the next time step, on average at most $\log_2 K$ bits of information can be propagated at each step. Moreover, given $\mathbf{h}_t$, the output, $\mathbf{v}_t$, at time t has no influence on the emission distribution at the next time step. The only way to make this model more powerful is to use a larger value of K and the returns are only logarithmic in K.

2.3.3 GMM-HMMs for Speech Recognition

A GMM-HMM is an HMM with GMM emission probability distributions, i.e. $p_i(\cdot; \boldsymbol{\theta}_i)$ is a GMM, as defined in equation 2.1, for each hidden state i. Given training data of sequences $\{\mathbf{V}_n\}$, with $\mathbf{V}_n = \mathbf{v}_1^n \cdots \mathbf{v}_{T_n}^n$, a GMM-HMM can be trained with the EM algorithm[3]. The application of GMM-HMMs to speech recognition is however much more involved than a straight forward application of EM because of the domain. We briefly outline some of these factors, and solutions to them.

In order to use a GMM-HMM for speech recognition we must define an appropriate hidden state space for the HMM. The state space should allow all possible words and word sequences to be modelled. In addition, each sequence of hidden states should ideally map to a unique sequence of words (i.e., the transcript)[4]. A naïve way of achieving this would be to map each phoneme to a unique hidden state. However, an order $p = 1$ Markov Model would allow many more phoneme sequences than the ones that correspond to words in a dictionary. A better approach would be to map each *word* to a unique hidden state. However, a single hidden state per word would not be a good model of the data because it would model the acoustics of a word as a mixture model with no sequential structure. Instead, each word could be modelled as a sequence of hidden states - this would capture the sequential structure to the extent that is possible with HMMs. However, this approach still has the limitation that it requires training data for each word in the dictionary. In addition, it uses no information about the units of sounds - phonemes - that are shared among different words. The solution that the speech community has evolved is to build an HMM for smaller units of sound. When the units are individual phonemes, the models is known as a *monophone* system; when the units are individual phonemes conditioned on their left and right contexts, they are known as context-dependent phoneme models. The most successful speech systems typically use *triphone* models - an HMM is used to model the acoustics for the central phone of a sequence of three phones. HMMs are usually modelled as left to right HMMs, i.e. transitions are only possibly from one state to itself or the next state, to its right. Typically 3 states are used — an incoming state, a middle state and an outgoing state — and the HMMs are referred to as tri-state HMMs. The incoming state and the outgoing state are used to model the high variability acoustic regions resulting

[3]Note that each sequence $\mathbf{V}_n$ may be of a different length, T_n

[4]Homophones would of course violate this assumption. Fortunately, language models can often select the most likely transcripts from a set of homophones.

from transitioning from one sound to the other while the middle state is used to model the less variable acoustic region corresponding to the center of the pronunciation of a phone. An HMM for a transcript is constructed by stringing together HMMs of individual units that make up the sequence of words ([Young et al., 2002]). For example, using the phonetic code from the TIMIT database, the transcript 'Hello World' would have the pronunciation (ignoring silence between the words): *hh eh l ow w er l d*. The HMM for the utterance would be a left to right HMM created by concatenating HMMs defined for the phones *hh, eh, l, ow, w, er, l,* and *d*. The end outgoing state of one phone is connected to the incoming state of the next phone in the sequence. Given a training corpus, a different HMM would be created for each utterance using this procedure. However all the HMMs share the parameters of the HMMs corresponding to the individual units (monophones in this case). This way of training the models is often referred to as embedded training [Young et al., 2002]. At test time, a combined HMM is created from all possible word HMMs and the language model is used to compute transition probabilities between terminal states of one word HMM and the start state of another word HMM.

The other main step in using a GMM-HMM for speech recognition is training the GMM-HMM parameters. Transition probabilities between word boundaries are dictated by the language model and are fixed. The transition probabilities within the states of an HMM are also usually fixed (to uniform values) - tuning these has little impact on the performance of speech recognizers and may in fact hurt the performance of the trained system (personal communication with Vincent Vanhoucke). Thus, training the GMM-HMM model primarily involves training the GMM parameters. Training of monophone models requires almost no tweaks to the EM algorithm. Training GMM's for triphone models however requires several techniques to deal with the large number of HMMs that arise from the large number of unique triphones. Most of these techniques involve parameter sharing between different states and HMMs. An understanding of these techniques is not essential for this thesis so we refer readers to other sources [Gales, 1999, Povey et al., 2011a, Gales, 1998, Young et al., 1994, Bahl et al., 1991, Saon et al., 2004]. In addition, a variety of techniques, referred to as speaker adaptive training (SAT), have been developed to adjust for mismatch between testing and training environments and speakers [Lee and Rose, 1998, Wegmann et al., 1996, Leggetter and Woodland, 1995, Padmanabhan et al., 2000].

2.4 DNN-HMM Speech Recognizers

Hybrid models of Neural Networks and Hidden Markov models were introduced more than two decades ago [Morgan and Bourlard, 1990, 1995, Renals et al., 1994, Bengio et al., 1992]. These models start with a GMM-HMM and attempt to improve the model by using a neural network to predict the posterior probabilities of the states from the observations.

Specifically, given an observation sequence $\mathbf{V} = \mathbf{v}_1\mathbf{v}_2\cdots\mathbf{v}_T$ and its transcript $\mathbf{W} = w_1w_2\cdots w_L$, a GMM-HMM model $p_{\mathbf{W}}(\cdot;\boldsymbol{\Theta}_g)$ is created from a trained GMM-HMM model $p_{\mathbf{G}}(\cdot;\boldsymbol{\Theta}_g)$ using the concatenative / embedding technique described above (i.e., we chain together the GMM-HMM units that correspond to the utterance). $p_{\mathbf{W}}(\cdot;\boldsymbol{\Theta}_g)$ is used to generate the sequence $s_1s_2\cdots s_T$ that has the highest probability of generating $\mathbf{V}$ (i.e. $s_1s_2\cdots s_T$ is the Viterbi sequence for the observation). A neural network $\mathbf{N}$ is then trained to predict the Viterbi state s_t that corresponds to observation $\mathbf{v}_t$ [5]. Once the Neural Network is trained it is used at test time instead of the GMM to perform *decoding*[6].

[5] In practice, contiguous frames $\mathbf{v}_{t-W}\cdots\mathbf{v}_t\cdots\mathbf{v}_{t+W}$ of features are used to predict the state of the center frame, s_t. We use $W = 7$ in thesis.

[6] Decoding is the term the speech community uses to refer to finding the best word sequence corresponding to an

The justification offered for this procedure is that the Neural Network can be used to improve upon the emission distribution of the GMM-HMM. By Bayes Rule,

$$p_{\mathbf{W}}\left(\mathbf{v}_t|s_t;\mathbf{\Theta}_g\right) = \frac{p_{\mathbf{W}}\left(s_t|\mathbf{v}_t;\mathbf{\Theta}_g\right)p_{\mathbf{W}}\left(\mathbf{v}_t;\mathbf{\Theta}\right)}{p_{\mathbf{W}}\left(s_t;\mathbf{\Theta}_g\right)} \tag{2.3}$$

During decoding we are searching for the state sequence $s_1 s_2 \cdots s_T$ that maximizes the probability:

$$p_{\mathbf{W}}\left(s_1 s_2 \cdots s_T|\mathbf{V};\mathbf{\Theta}_g\right) = \frac{p_{\mathbf{W}}(\mathbf{V}|s_1 s_2 \cdots s_T;\mathbf{\Theta}_g)p_{\mathbf{W}}(s_1 s_2 \cdots s_T;\mathbf{\Theta}_g)}{p_{\mathbf{W}}(\mathbf{V};\mathbf{\Theta}_g)} \tag{2.4}$$

Here, $p_{\mathbf{W}}\left(\mathbf{V};\mathbf{\Theta}\right)$ can be ignored because it is fixed. By using a model of the posterior $p_N\left(s_t|\mathbf{v}_t\right)$ that is better suited to the data than $p_{\mathbf{W}}\left(s_t|\mathbf{v}_t;\mathbf{\Theta}_g\right)$ we can improve on the emission distribution of the GMM-HMM $p_{\mathbf{W}}\left(\mathbf{v}_t|s_t;\mathbf{\Theta}_g\right)$. This improves $p_{\mathbf{W}}(\mathbf{V}|s_1 s_2 \cdots s_T;\mathbf{\Theta}_g) = \prod_{t=1}^{T} p_{\mathbf{W}}\left(\mathbf{v}_t|s_t;\mathbf{\Theta}_g\right)$ which leads to better decoding.

Although they were introduced a long time ago the effectiveness of these models was compromised by two main factors. Firstly, the models did not use multiple layers of hidden units, typically using only one layer [Ellis and Morgan, 1999, Bourlard et al., 1992]. Even when multiple layers were used the number of units in each layer was too small [Albesano et al., 2000]. Only recently have these techniques been applied with a large number of hidden units and with architectures using 6 or 7 layers [Hinton et al., 2012a]. The advent of GPUs has made the exploration of such architectures and their eventual use in speech recognizers possible. Secondly, new techniques that allow deep neural networks to be trained more effectively have only recently been developed [Hinton and Salakhutdinov, 2006].

Recently, there has been widespread adoption of these models in speech recognizers [Hinton et al., 2012a]. These models typically use neural networks with 6 or more layers of units, and depth is an integral part of the recipes as our results later in tables 2.2 and 2.3 show. To underscore these differences the community now refers to these techniques as Deep Neural Network - Hidden Markov Models (DNN-HMM).

Nevertheless, it would be incorrect to say that each of these ingredients was not attempted before. In fact, each of the factors - depth of the network, the number of hidden units in a layer and context dependent models - had been experimented with individually with promising results. For example:

- Albesano et al. [2000] used an ANN with 2 layers of hidden units to predict *Stationary-Transitional Units (STU)*[7] for use in an ANN-HMM system.

- Ellis and Morgan [1999] showed how a larger number of hidden units and larger amount of training data improves the performance of an ANN-HMM system with one hidden layer of units.

- Cohen et al. [1992], Franco et al. [1994] and Bourlard et al. [1992] applied ANN-HMM systems to context dependent ANN-HMMs with strong gains over context dependent GMM-HMMs and context independent ANN-HMMs.

observation - i.e. finding the *maximum a posteriori* (MAP) word sequence from the model.

[7] These are di-phone like units

dataset	Error Rate on dev / test set
TIMIT	32.7 / 33.0 %
WSJ (81 hours) - Speaker independent testing	12.7 / 8.3%
WSJ (81 hours) - Speaker dependent testing	9.5 / 5.79%

Table 2.1: Error rates of GMM-HMM baselines used in this thesis. For TIMIT, the Phone Error Rate (PER) was used since this is a phone recognition dataset. Word Error Rate (WER) was used for WSJ. The WSJ baselines were created with the si-284 dataset, which has acoustic data for 81 hours. The baseline was created with the speaker adaptive recipe *tri4b*, available in Kaldi. Note that the test set and dev set data were developed for different tasks and *test-eval92* is easier - hence has a lower WER.

2.5 Baselines

In this section we describe the baselines we use in the thesis. We used Kaldi to build GMM-HMM models [Povey et al., 2011b]. See table 2.1 for a summary of GMM-HMM baselines. These models were used to generate the training data for the DNN-HMMs by forced alignment [8]. Baseline DNN-HMM models were then trained using MFCC and FBANKs inputs.

2.5.1 TIMIT

For TIMIT, a GMM-HMM model was created from a slightly modified version of recipe *s3* in Kaldi (see appendix for information on the recipe). The HMM model has 180 states.

DNNs with two to seven layers of hidden units were trained to predict the 1 of 180 states for each time frame, using a context of +/ 7 frames, i.e. 15 frames of inputs were used to predict HMM states. Previous studies have shown that this window size leads to high accuracy in phone recognition ([Mohamed et al., 2012]). Once the DNN's were trained they were used to predict the log of posterior probabilities of states for the frames of the validation (dev) and test sets. These were then decoded using Kaldi, and matched to the actual transcripts to get the Phone Error Rates (PER).

Figure 2.2 and tables 2.2, 2.3 show a summary of the results obtained by running the neural network models for three different runs using the alignments obtained from the GMM-HMM baseline and the DNN-HMM baseline. It can be seen the FBANKs outperform MFCCs and the difference between the two increases with deeper models.

2.5.2 Wall Street Journal

A GMM-HMM model was created for WSJ from recipe *s5* in Kaldi. The model *tri4b* was used to generate the forced alignments. This model was trained on all 81 hours (commonly referred to as *si-284*) of WSJ with speaker-adaptive training. For the thesis, for the DNN-HMM systems we used the 14 hour subset *si-84*. Since our goal is to compare performance of different features, we felt that the 14 hour subset would offer a representative view of the effect of using our techniques, while allowing us to run enough experiments in a reasonable time frame.

The recipe generated GMM-HMMs with 3385 states. Baseline neural networks were trained to predict the states for each time frame using a context of +/- 7 frames. The predictions from the trained DNN's were decoded with Kaldi and matched to the actual transcripts to get the Word Error Rates. As the WSJ dataset was much larger than TIMIT, we used only 6 layers of hidden units for our baselines. Table 6.3 shows a summary of the word error rates (WER) obtained by running the neural network models

[8] Forced alignment refers to generating the best path that corresponds to the correct transcript.

# of layers	MFCC		FBANKS	
	dev	test	dev	test
2	21.9,22.1,22.0(**22.0**)	23.6,23.8,23.6(**23.7**)	20.3,20.1,20.0(**20.1**)	22.1,22.1,22.2(**22.1**)
3	20.6,21.0,21.2(**20.9**)	22.4,22.8,23.1(**22.8**)	19.6,19.4,19.7(**19.6**)	21.6,21.8,21.6(**21.7**)
4	21.4,21.4,21.2(**21.3**)	23.1,23.0,22.7(**22.9**)	19.5,19.5,19.4(**19.5**)	21.3,21.9,21.6(**21.6**)
5	20.8,21.1,21.3(**21.1**)	23.0,22.7,22.7(**22.8**)	19.4,19.9,19.6(**19.6**)	21.0,21.3,21.6(**21.3**)
6	20.8,20.8,20.8(**20.8**)	22.7,22.8,22.2(**22.6**)	19.6,19.6,19.6(**19.6**)	21.3,20.5,21.3(**21.0**)
7	20.6,20.8,21.0(**20.8**)	22.3,22.3,22.7(**22.4**)	19.1,19.4,19.3(**19.3**)	21.2,21.7,21.0(**21.3**)

Table 2.2: PER of DNN-HMMs baselines on TIMIT. We trained neural networks with two to seven hidden layers, each containing 2000 sigmoid units. The input to the neural network was 15 frames of features (MFCC or FBANK) augmented with deltas and accelerations. Each configuration was trained three different times. The mean for each combination is in shown in bold, along with the values for the three runs.

# of layers	MFCC		FBANKS	
	dev	test	dev	test
2	21.3,21.3,21.0(**21.2**)	23.6,23.0,23.1(**23.3**)	19.7,19.9,19.8(**19.8**)	21.7,21.9,21.9(**21.8**)
3	21.0,20.9,20.8(**20.9**)	22.5,23.0,23.0(**22.9**)	19.5,19.6,19.5(**19.5**)	21.1,21.0,21.4(**21.2**)
4	20.7,21.1,20.7(**20.8**)	22.1,22.7,22.2(**22.3**)	19.1,19.1,19.2(**19.1**)	21.1,20.8,21.2(**21.0**)
5	20.6,20.5,20.4(**20.5**)	22.3,22.6,22.3(**22.4**)	19.1,19.1,19.3(**19.2**)	20.8,20.7,21.0(**20.8**)
6	20.7,20.6,20.3(**20.5**)	22.5,22.5,22.4(**22.5**)	19.0,19.2,19.2(**19.1**)	20.8,21.2,20.7(**20.9**)
7	20.6,20.4,20.4(**20.5**)	22.6,21.9,22.1(**22.2**)	19.1,19.2,19.1(**19.1**)	20.6,20.9,20.7(**20.7**)

Table 2.3: PER of DNN-HMMs systems on TIMIT after regenerating alignments using a three layer neural network trained on GMM-HMM alignments for eight epochs. These experiments used the same hyper-parameters as the experiments in table 2.2.

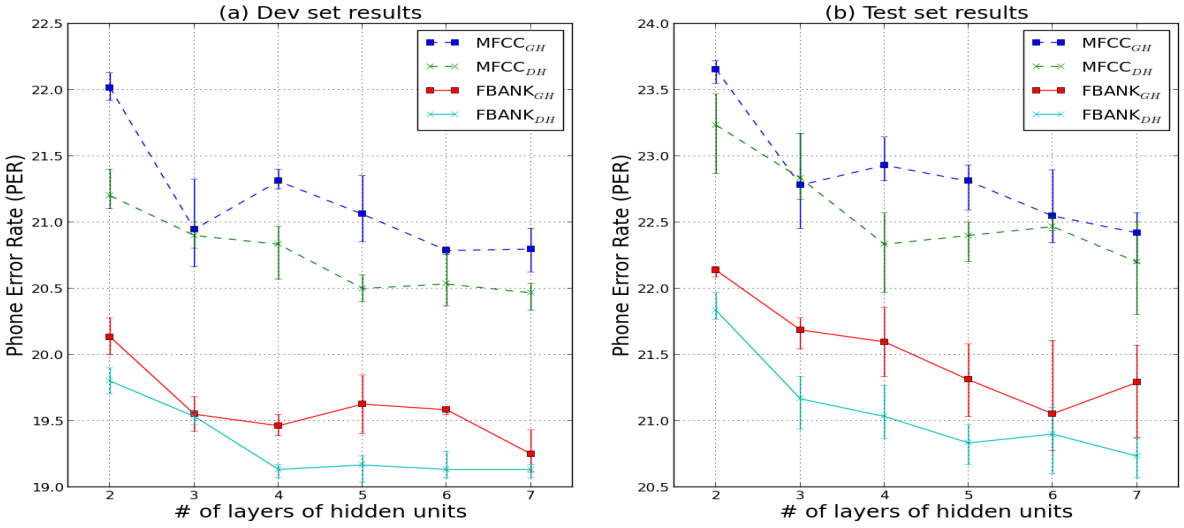


Figure 2.2: Effect of number of layers of hidden units on Phone Error Rate (PER) for TIMIT on (a) the dev set and (b) the test set. A GMM-HMM model was used to generate alignments to train the first set of DNN-HMM models (models labelled MFCC$_{GH}$ and FBANK$_{GH}$ in the figures). A three layer DNN-HMM model with FBANK inputs was used to generate alignments for the second set of DNN-HMM models (models labelled MFCC$_{DH}$ and FBANK$_{DH}$ in the figures). DNN-HMM models with FBANK inputs have higher accuracy (lower PER) than DNN-HMM models with MFCC inputs. It can be seen that regenerating alignments improves the results, and also leads to a cleaner trend with depth.

# of layers	MFCC				FBANK			
	run 1	**run 2**	**run 3**	**average**	**run 1**	**run 2**	**run 3**	**average**
6	9.7/6.0	9.7/5.8	9.9/5.6	9.7/5.8	9.3/5.3	9.2/5.4	9.6/5.4	9.4/5.4

Table 2.4: WER of DNN-HMMs system on WSJ 14 hour subset (*si-84*). Each entry reports the dev set / test set WER for a run.

for three different runs. The alignments from the GMM-HMM system were used for all experiments and not regenerated using the DNN-HMM system, like we did for TIMIT.

We note that each experiment in this thesis was performed in triplicate to get a sense for the variability introduced by neural network training. The mean values are reported in each table. However, we did not perform a statistical assessment of the results because such an assessment would be inherently flawed — neural network training results can be highly variable based on the hyperparameters. However, Deep Belief Network (DBN) pretraining of parameters mitigates this variability to a large extent because of the generative pre-training of the weights. We used the same hyper-parameters throughout the thesis for all the neural network training related to the frame level classifications (unless pointed out). We point out that the hyperparameters used in the baseline have been strongly influenced by prior work, but the features learnt in this thesis are entirely novel. In that respect, it is possible that each of the experiments in the subsequent chapters may have performed better with more tuning of hyper-parameters. The triplicate experiments are intended to provide some qualitative assessment of the variability in each of these experiments.

Part II

Unsupervised Feature Learning

Chapter 3

Learning Features From a Generative Model of Raw Speech Signals

Speech sound waves have complex non-stationary distributions that have long evaded statistical modelling at the level of raw signals. In the early days of speech processing, recognition and generation of speech signals were treated as related problems and production models of speech signals were developed with the intention of using them for speech recognition [Stevens et al., 1953, Flanagan and Cherry, 1969, Flanagan, 1972, Flanagan et al., 1970, Ishizaka and Flanagan, 1972, Coker and Fujimura, 1966]. However these models aim to provide a physics-based generative model of speech and require a detailed specification of the physical system (articulatory apparatus) and environment (acoustic conditions of the room) to be accurate. Statistical modelling of raw speech signals on the other hand aims to build an observational generative model of speech signals. Until recently, statistical modelling of speech signals was relegated to the use of GMM-HMM to model lossy, frame-based representations of speech such as coefficients from PLC, DFT and MFCC. For example Poritz [1982] and Sheikhzadeh and Deng [1994] use HMMs to model raw speech signals using linear prediction polynomials and their variances. Inherently the models make strong assumptions about speech signals within a frame (for example, that linear autoregressive models of order six are sufficient to capture important aspects of the signal) that are not necessarily true. Given the amount of speech signal that is available, learning models of raw speech signals from data alone seems like a promising alternative.

Recently, Independent Components Analysis (ICA) has been used to learn features from raw speech signals [Lee et al., 2000]. [Lee et al., 2000] learn features on decorrelated speech segments of length 3.1 ms sampled at 16 kHz (i.e., 50 samples long). These features outperformed MFCCs on a word spotting task in Korean. Lewicki [2010, 2002] and Smith and Lewicki [2006] explored a similar idea, using a probabilistic generative model. They showed that a sparse, independent prior on latent variables led to discovery of Gammatone like features. The frequency selectivity and response of these features is very similar to that observed in auditory fibers of cats [Carney and Yin, 1988, Greenberg et al., 1986]

Since then, raw speech signals have been used as inputs to a convolutional network that feeds its predictions to a Conditional Random Field for speech recognition [Palaz et al., 2013]. However the results

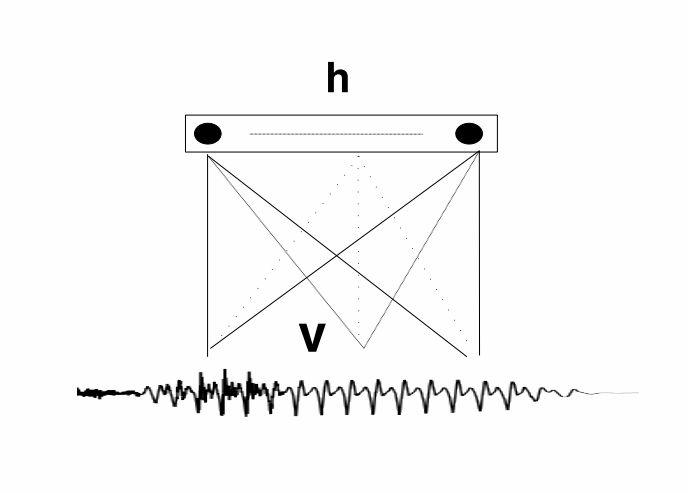


Figure 3.1: RBM is used to model fragments of speech signals.

achieved were not as good as those achieved in Mohamed et al. [2010] using a Conditional Random Field that received input from MFCC features.

In this work we developed a generative model of raw speech using a Restricted Boltzmann machine (RBM) [Smolensky, 1986]. We show that by learning the standard deviations of the visible units of a Gaussian - Rectified Linear Unit (ReLU) RBM [Nair and Hinton, 2010] we can discover meaningful Gammatone like features. We then show that these features can be used in a speech recognition system to provide better performance than MFCCs. However, these features typically under-perform log Mel filter banks on speech recognition tasks.

3.1 Probabilistic Model

Let $\mathbf{v}$ represent a fragment of speech signal of length 80 samples representing 5 ms at 16 kHz. We model the probability distribution of such fragments on the TIMIT corpus using an RBM which is an energy based model that has hidden (latent) variables $\mathbf{h}$, and visible (observed) variables, $\mathbf{v}$ (see figure 3.1). Each joint configuration of these variables is assigned an energy, $E(\mathbf{v}, \mathbf{h})$, and the probabilities of joint configurations are defined by a Boltzmann distribution:

$$p(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} e^{-E(\mathbf{v}, \mathbf{h})} \tag{3.1}$$

where Z is the partition function $Z = \sum_{(\mathbf{v}, \mathbf{h})} \exp(-E(\mathbf{v}, \mathbf{h}))$.

For a Gaussian-Bernoulli RBM, the visible variables, $\mathbf{v}$, are real valued, and the hidden variables, $\mathbf{h}$, are binary. The energy of a joint configuration $(\mathbf{v}, \mathbf{h})$ is:

$$E(\mathbf{v}, \mathbf{h}; \boldsymbol{\Theta}) = -\mathbf{v}^{\mathbf{T}}\mathbf{W}\mathbf{h} - \mathbf{h}^{\mathbf{T}}\mathbf{a} + \frac{1}{2}(\mathbf{v} - \mathbf{b})^T \boldsymbol{\Sigma}^{-1}(\mathbf{v} - \mathbf{b}) \tag{3.2}$$

where the matrix $\mathbf{W}$ represents interaction strengths between different visible variables (rows) and hidden variables (columns). $\mathbf{b}$ and $\mathbf{a}$ represent the visible and hidden biases, $\boldsymbol{\Sigma} = diag(\boldsymbol{\sigma})^2$ represents the diagonal (not necessarily spherical) covariance matrix of visible units conditioned on the hidden units and $\boldsymbol{\Theta} = \{\mathbf{W}, \mathbf{b}, \mathbf{a}, \boldsymbol{\sigma}\}$ is the vector of all the parameters.

The layered structure of RBMs makes it easy to do fast sampling using block Gibbs [Geman and Geman, 1984, Smolensky, 1986, Liu et al., 1994]. Conditioned on visible variables $\mathbf{v}$, the hidden variables have an independent Bernoulli distribution with probability vector $\sigma\left(\mathbf{W}^T\mathbf{v} + \mathbf{a}\right)$, where the sigmoid is applied element-wise. Conversely, given hidden variables $\mathbf{h}$, the visible variables have a Gaussian distribution given by $\mathcal{N}(\mathbf{\Sigma W h} + \mathbf{b}; \mathbf{\Sigma})$.

It is possible to replace an individual binary unit by an infinite number of binary variables coupled together with the same incoming weights but with biases, $\mathbf{a}$, stepped downwards by 1 (starting at -0.5). It was empirically shown in Nair and Hinton [2010] that such a coupled set of binary variables, called **S**tepped **S**igmoid **U**nits (SSU), has the property that the distribution of the sum of activities of the coupled units can be closely approximated by a rectified linear unit (ReLU) whose value is $max(0, N(x, \sigma(x)))$, where x represents the input into each of the binary variables (without addition of any bias) and $\sigma(x)$ is the logistic sigmoid function[1]. Another way of looking at this sum is as an approximation to the area under the sigmoid function over the interval $(x - M, x)$, i.e.:

$$\lim_{n\to\infty} \sum_{i=0}^{Mn} \frac{1}{n}\sigma\left(x - \frac{i}{n}\right) = \int_0^M \sigma\left(x - y\right) dy = \log\left(1 + \exp\left(x\right)\right) - \log\left(1 + \exp\left(x - M\right)\right) \tag{3.3}$$

As M goes to infinity the second term on the right approaches 0 and we only have the soft-plus function. The variance of the sum under a Bernouilli distribution at each point is $\sigma\left(x\right)$, as can be seen by taking the derivative w.r.t. x on both sides - the left hand side transforms into the sum of variance of the individual Bernoullis $(\sigma\left(x - \frac{i}{n}\right)\left(1 - \sigma\left(x - \frac{i}{n}\right)\right))$ and the right side transforms into $\sigma\left(x\right)$. In this chapter, we used an RBM with ReLU hidden units and Gaussian visible units - a Gaussian-ReLU RBM. For a given visible variable $\mathbf{v}$ the hidden units are distributed as:

$$p(\mathbf{h}|\mathbf{v}) = \mathcal{N}\left(\log\left(1 + \exp\left(\mathbf{W}^T\mathbf{v} + \mathbf{a}\right)\right); diag\left(\sigma\left(\mathbf{W}^T\mathbf{v} + \mathbf{a}\right)\right)\right) \tag{3.4}$$

where the log, exp and sigmoid are applied element-wise to the vectors and $diag\left(\cdot\right)$ constructs a diagonal matrix from an input vector by placing the elements of the vector on the diagonal. The distribution of the visible variables given the hidden variables is the same as that given above for the Gaussian-Binary RBM.

3.2 Learning

The gradient of the log of the probability of the training data w.r.t. a weight parameter, $w_{ij} \in \mathbf{W}$ is as follows [Hinton, 2002]:

$$\frac{\partial}{\partial w_{ij}} \log p(\mathbf{v}; \mathbf{\Theta}) = < v_i h_j >_{data} - < v'_i h'_j >_{model} \tag{3.5}$$

where $<>_{data}$ denotes the expectation under the distribution of $\mathbf{h}$, conditioned on the data vector, $\mathbf{v}$, and $<>_{model}$ denotes the expectation under the models' distribution over all joint configurations $(\mathbf{v}', \mathbf{h}')$. In this paper, we were also interested in learning the standard deviations $\boldsymbol{\sigma}$ of the units. The gradient

[1] We are using σ for both standard deviations and the sigmoid function in this chapter - the sense in which it is applied can easily be inferred from the context.

of the log probability of the data w.r.t. $a_i = \log \sigma_i$ is as follows:

$$\frac{\partial}{\partial a_i} \log p(\mathbf{v}; \mathbf{\Theta}) = \frac{(v_i - b_i)^2}{\sigma_i^2} - < \frac{(v_i' - b_i)^2}{\sigma_i^2} >_{model} \tag{3.6}$$

We used the "Contrastive Divergence" (CD) algorithm [Hinton, 2002] for learning the parameters with stochastic gradient descent (SGD) [Bottou, 1998]. This algorithm has been applied successfully to learning parameters of different types of RBM's on different datasets [Taylor et al., Memisevic and Hinton, 2007, Welling et al., 2004, Roth and Black, 2005]. We used CD-1 for this thesis.

3.2.1 RBM Training

The entire training data set (TIMIT or WSJ) was normalized to unit standard deviation. RBMs with different numbers of visible and hidden variables were trained. In each case, the RBM was initialized with weights drawn from $\mathcal{N}(0, \frac{1}{n_u})$ where n_u is the number of visible variables. The hidden and visible biases were initialized to 0 and standard deviations of visible units were initialized to 1. Stochastic gradient descent was performed using mini-batches of 128 randomly selected windows of speech (which were thus in any random phase with respect to the start of the parent sentences). We used a momentum of 0.5 and a learning rate of 0.01 on average gradients. An L1-penalty of 0.001 was used. The learning rate was annealed as $1 - (t/T)^{0.8}$ where T is the total number of epochs of training and $t = 0 \cdots (T-1)$ was any given epoch. Every time the reconstruction error on the validation set went up at the end of an epoch, compared to the last epoch, the learning rate was dropped to half of its value in the previous epoch; this halving of learning rate was on top of the annealing scheme above. The training time was chosen so that each point in the training set was sampled 30 times, on average.

3.2.2 RBM Results

Figure 3.2 shows the weights that were learnt from the TIMIT database using 120 hidden variables and 80 visible variables. Each subplot corresponds to the weights connecting a hidden unit to the 80 visible variables. Since each unit responds maximally to input that looks exactly like the incoming weights, we can infer their frequency selectivity by looking at the Fourier transform of the incoming weights. Each filter looks very much like a Gammatone and has a center frequency that it responds to the most. In the figure we have ordered the plots in order of increasing center frequency. Figure 3.3 shows the features learnt from WSJ database si-84, using the same configuration. It can be seen that these are very similar in nature to the features learned from TIMIT. In fact the features learned by ICA in Lee et al. [2000] and Lewicki [2002] are very similar to these but use a different type of prior on features. Those methods assume that the features are marginally independent, but observations make them conditionally dependent; the RBM on the other hand assumed features are marginally dependent but conditionally independent when the visible variables are given. Interestingly, in both cases almost every feature (specifically the ones with smaller center frequencies) matches up with another feature that is the negative of itself. For example, see feature pairs 1 and 3, 4 and 5, 7 and 10, 11 and 12, and 9 and 13 (starting at 1) in the first row of figure 3.2. This is because speech patterns are symmetric around 0 (almost) and so each pattern and its negative are equally likely, but an individual ReLU feature can only capture one of these patterns — the other would have a negative response and would be rectified. Have negatively matching features fixes this problem. Table 3.1 shows the root mean squared (RMS)

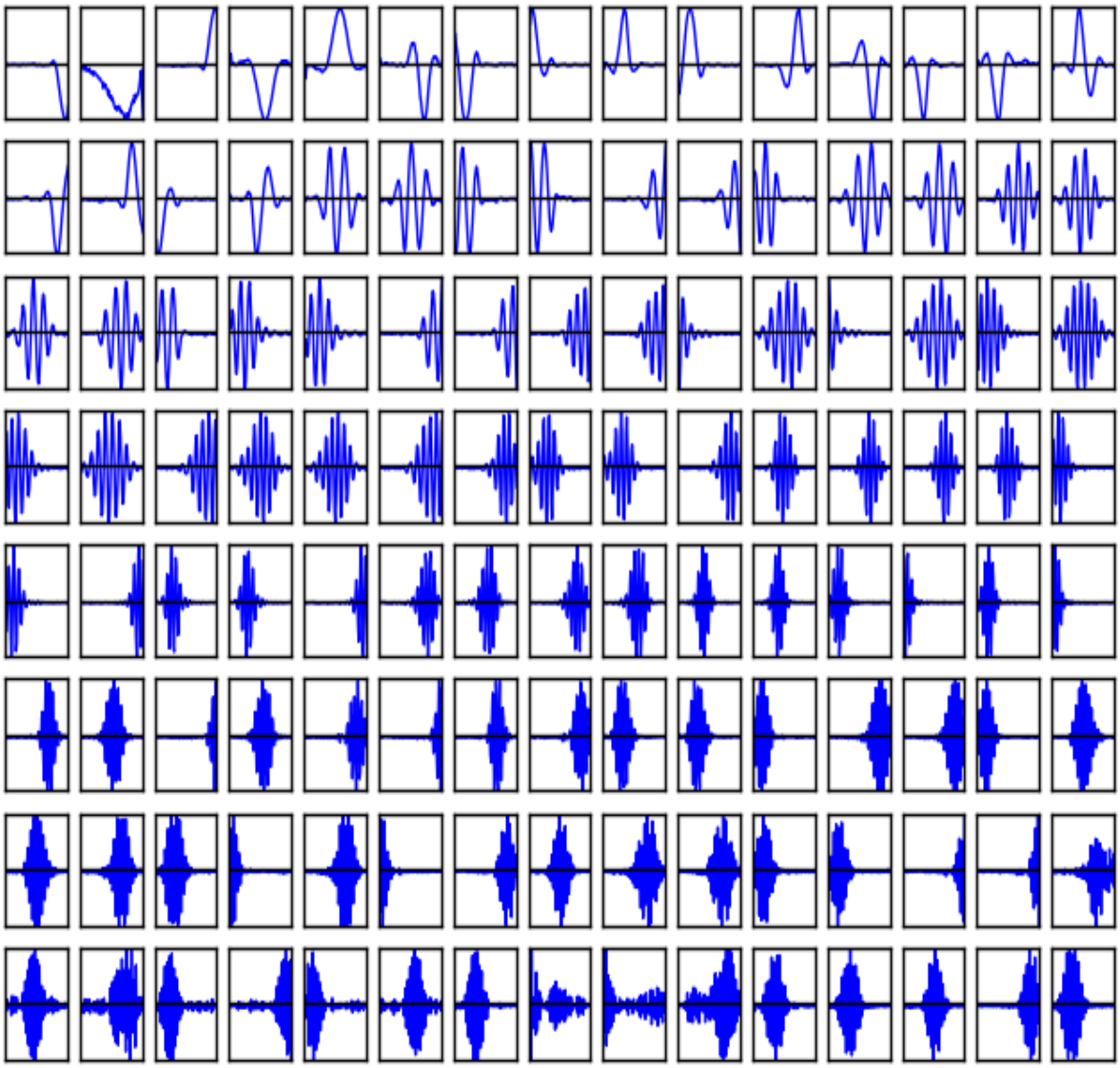


Figure 3.2: Features learnt from TIMIT training data using a window size of 5ms (80 samples) with 120 hidden units. Each plot shows the incoming weights from the data into a hidden unit of the RBM. The filters are sorted in order of increasing center frequency of the incoming weights. The black line is the x-axis.

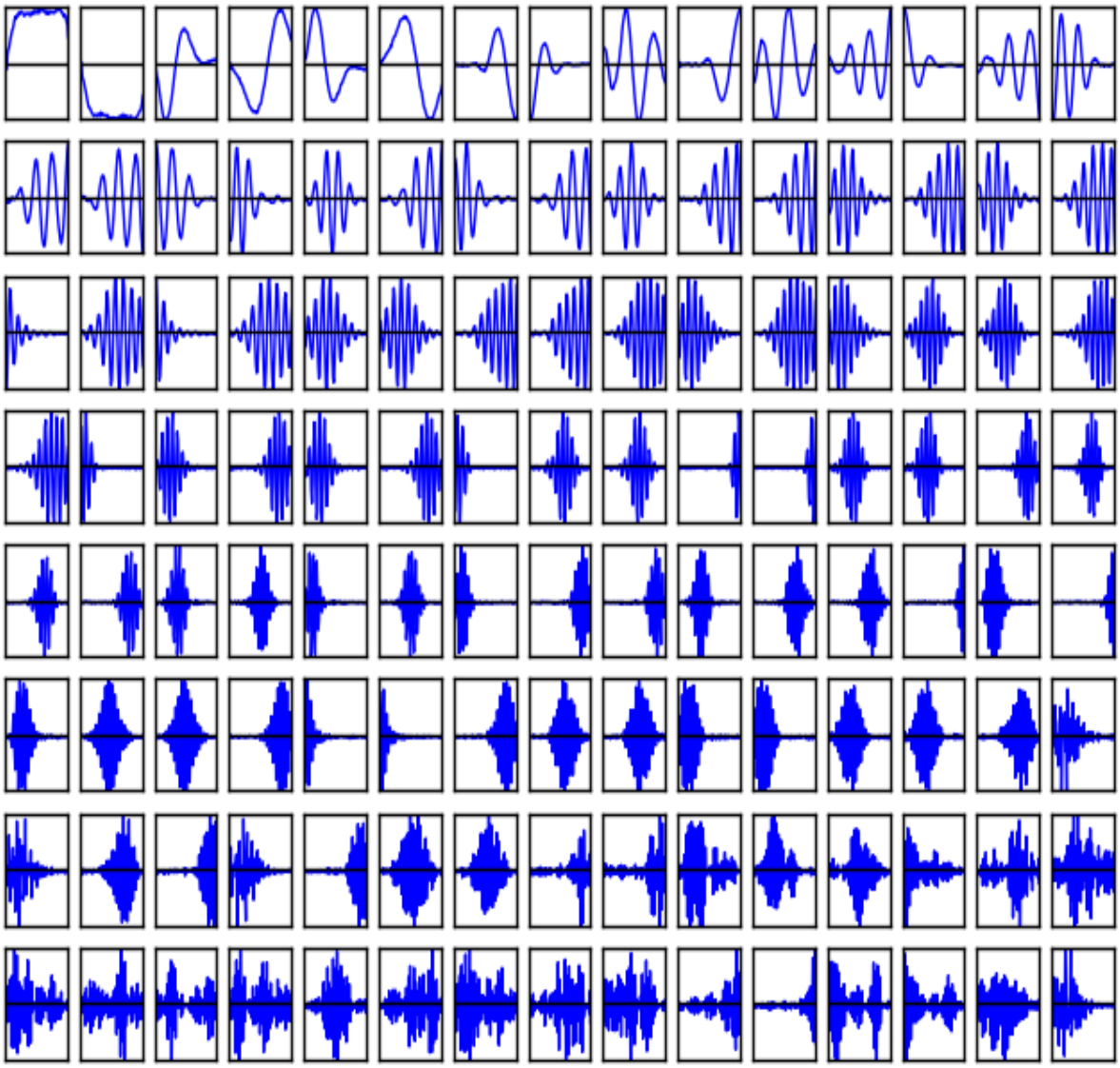


Figure 3.3: Features learnt from WSJ training data (si-84) using a window size of 5ms (80 samples) and 120 hidden units. See figure 3.2 for details.

window	# of Units		
	80	120	160
80	0.171	0.1185	0.077
160	0.264	0.216	0.176
320	0.384	0.296	0.295

Table 3.1: Table showing Root Mean Squared (RMS) reconstruction error (per sample) for different window sizes, and number of units. Using more units, while keeping the window size the same reduces the reconstruction error per sample. Using a larger window with the same number of units increases the reconstruction error per sample. Note that using the same ratio of window size to number of units keeps the reconstruction error per sample about the same. For example the RMS reconstruction error for the case where the number of units and the window size are both 80 is 0.171; for the case where they are both 160, the RMS reconstruction error is 0.176. The slight increase is probably because larger sequences are inherently more difficult to model.

difference between dimensions of data vectors and their mean field reconstructions using different input window sizes and numbers of hidden units. The mean field reconstruction of a data vector $\mathbf{v}$ is computed in two steps. First the data is used to compute the mean value of hidden units using equation 3.4. This is then taken as the value of the hidden units $\mathbf{h}$ and used to compute the conditional mean of the visible variables which is the reconstruction. It can be seen from the table that reconstruction error generally decreases with the number of hidden units, but increases with the window size. Figure 3.4 shows the raw signal for an utterance from the TIMIT test set in panel (a) and its reconstruction with an RBM with 120 hidden variables and 80 visible variables in panel (b). The reconstruction is made by averaging overlapping mean field reconstructions of frames speech signals. Each frame is 80 samples long (5 ms) and has an overlap of 40 samples (2.5 ms) with the the next segment; overlapping reconstructions are averaged. Panels (c) and (d) show the spectrogram of the original signal and the reconstruction. Its clear from panels (a)-(d) that the segments of speech corresponding to vowel sounds are well reconstructed - for example between 0.4-0.5 seconds and 0.6-0.9 seconds. However, the segments of sound corresponding to fricative like sounds is not well reconstructed - for example between 0.9-1.0 seconds and between 2.0-2.15 seconds. In fact the reconstruction seems to lose all the energy above 6000 Hz that is present in the original signal. This is because that part of the signal corresponds to turbulent flow and does not have a periodic signal. The input into the hidden units is a linear function of the signals, and is unable to cope with the aperiodic noise, and simply ignores this part of the signal. Panel (e) shows the mean values of the hidden units conditioned on the input signal. The hidden units are sorted in the same way as in figure 3.2 - from low center frequency to high center frequency sensitive hidden units. It can be seen that the activities of the features bear a strong resemblance to the spectrogram (below 6 kHz). Figures 3.5 and 3.6 zoom into a vocal and a fricative part of speech of figure 3.4[2]. Figure 3.5 shows that the reconstruction of the original signal is almost perfect. However, it also shows that very similar signals can have very different codes under this model. For example, each frame (separated by dashed lines) of the signal has a very similar pattern as the others - corresponding to the signal emanating from the mouth over the duration of one glottal pulse (opening and closing of the glottis). Nevertheless each frame has highly different activities of hidden units. This is because the model is not translationally invariant. Figure 3.6 shows that the model is unable to perform a very good reconstruction of the aperiodic / quasi-periodic signals. In the first frame the smooth, sinusoidal part of speech is well reconstructed, but the noisy segments thereafter are not well reconstructed. In fact the residual seems to have almost all

[2]For the zoomed in views, consecutive frames were not overlapped.

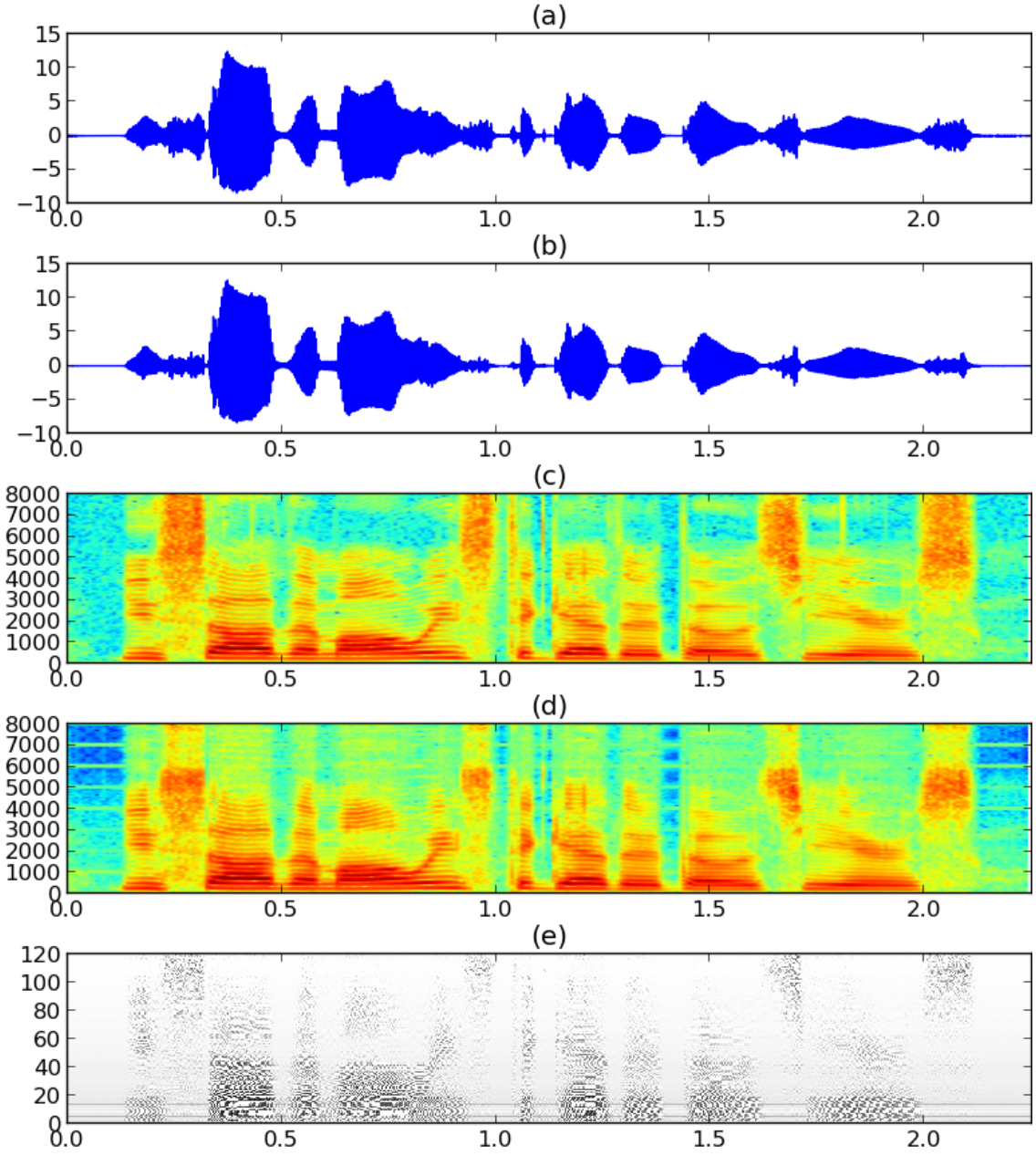


Figure 3.4: Reconstruction of an utterance with a Gaussian-ReLU RBM with 120 units trained on 80 samples of raw signal. The utterance is: "You saw them always together those years". (a) Original signal (b) Reconstruction created by reconstructing windows of size 80 sliding by 16 samples. (c) Spectrogram of original signal (d) Spectrogram of reconstruction. (e) Plot of the log of mean activities of the hidden variables of the RBM conditioned on the input signals. The hidden variables were sorted in order of increasing center frequency.

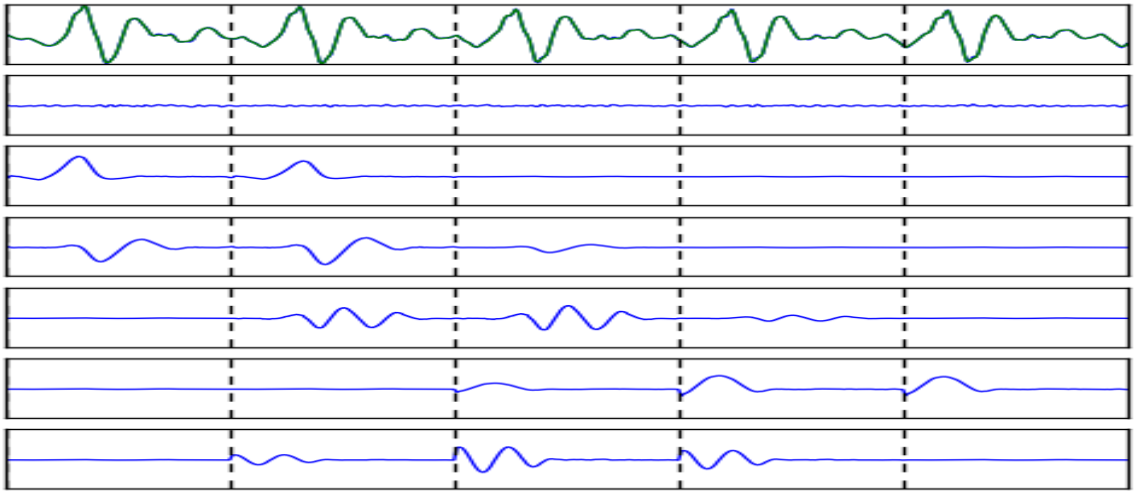


Figure 3.5: Zoomed in view of reconstruction in figure 3.4 between time 0.65 and 0.675 seconds. This segment corresponds to the vowel sound /ao/ (the vowel sound in the word 'bought'). Top Row: Original signal (blue) and reconstruction (green) are very similar. Second Row: Difference between reconstruction and original. Remaining Rows (Top to bottom): Contributions of the 5 features that contribute most in Euclidean norm of contribution (most contributing first). The y-axis for all plots were scaled to the same height (+/- 6). The dashed vertical lines show the location of the frame boundaries.

of the energy of the original signal.

We show samples generated from a learned model after 5000 blocked Gibbs steps in figure 3.7. Unfortunately, these samples do not appear to capture the distribution of the data very well. Gaussian-Binary RBM models that do not capture second order statistics of signals would be hard pressed at generating realistic samples. Other models, such as the mean-covariance RBM ([Ranzato and Hinton, 2010]), are better at capturing such structure but there is little evidence to suggest that better generative models lead to features that improve discriminative results compared to simple RBM models trained with CD-1.

3.3 Speech Recognition Experiments

The features we discovered above were used for speech recognition with a DNN-HMM hybrid system. Alignments that were generated for the MFCC and FBANK baselines in section 2.5 were used here as training data for neural networks.

Figure 3.8 shows how the features from the RBMs were used to compute inputs to the neural networks. Features were computed over windows starting at every sample of the raw signal - we loosely refer to this as a convolutional setup, although non-linearities may be applied. Thus mean hidden activities were computed 16,000 times for 1 second of speech. For each feature, the output values from contiguous frames were pooled over a window of 25 ms. Such pooled frames were created at a stride of every 10 ms. 15 frames of pool features were concatenated and used as input to predict the phoneme label of the middle frame. Each dimension of the input vector to the neural network was standardized to have

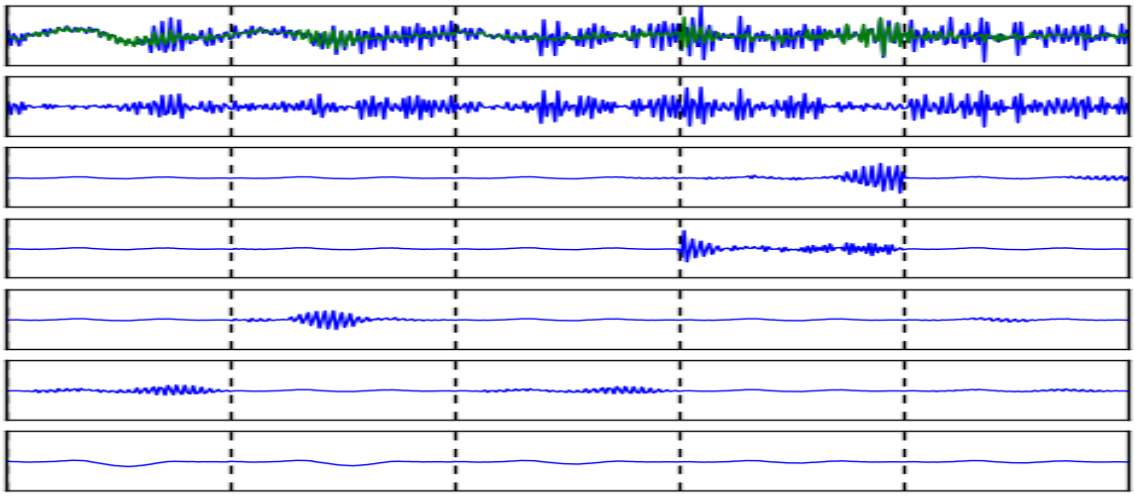


Figure 3.6: Zoomed in view of reconstruction in figure 3.4 between time 0.925 and 0.95 seconds. This segment corresponds to the fricative sound /z/ (The z sound in the word 'zone'). Top Row: Original signal (blue) and reconstruction (green). Second Row: Difference between reconstruction and original. Remaining Rows (Top to bottom): Contributions of 5 features the contribute most in Euclidean norm of contribution (most contributing first). All axes are scaled to the same height (+/- 3.5). The dashed vertical lines show the location of the frame boundaries. Clearly, the model is unable to perform a very good reconstruction of the parts of speech with turbulent flow.

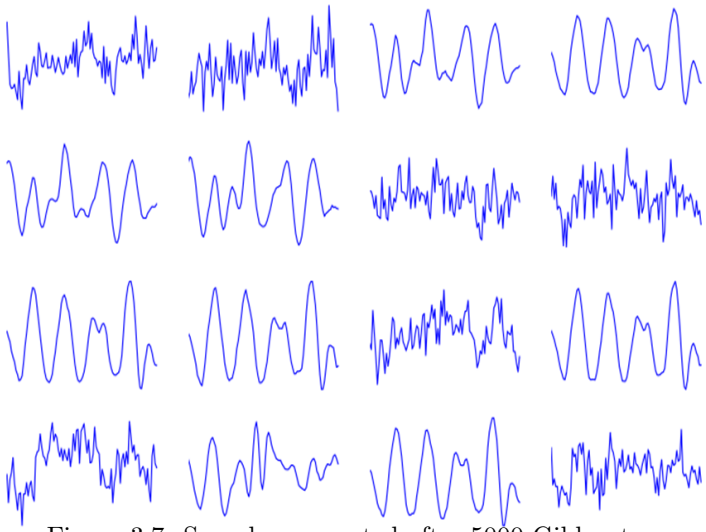


Figure 3.7: Samples generated after 5000 Gibbs steps.

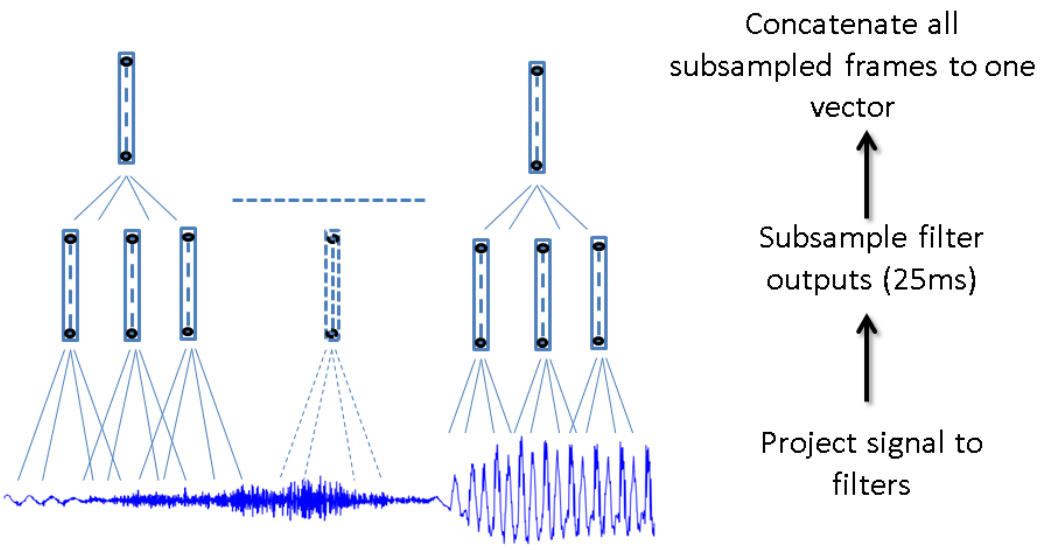


Figure 3.8: Convolutional setup used to create features for speech recognition.

a mean of 0 and a standard deviation of 1.

Table 3.2 considers the impact of different ways of computing the feature outputs from the RBM (column **activation**), pooling feature outputs (column **pool-type**), pooling size (column **pool-window**), and use of delta and acceleration vectors (column $\mathbf{\Delta, \Delta\Delta}$) on PER on dev and test sets of TIMIT. In each case a neural network with two hidden layers of 2000 sigmoid units each were trained to predict class labels for the input. Once trained these neural networks were used in a DNN-HMM hybrid model and phone error rates calculated after decoding of validation (dev) and test sets.

Three different pooling types were experimented with — $\log(\sum(\cdot))$, $\sum\log(\cdot)$ and $\sum(\cdot)$ where $\cdot$ refers to the values at the locations being pooled together. In $\log(\sum(\cdot))$, the outputs of the features were pooled (added) and then log-transformed. In $\sum\log(\cdot)$ pooling the outputs of the features were log transformed before pooling (adding). In sum pooling the outputs of the features were only added together. Among these different pooling types $\log(\sum(\cdot))$ leads to highest recognition accuracy. We also tried $(\sum\cdot)^{\frac{1}{3}}$ which has some support from perception research but the accuracy was on par with that obtained from $\log(\sum(\cdot))$ pooling.

We experimented with pooling windows and found that the standard window of 400 (25 ms) used for computing spectral features performed better than shorter pooling window of 160 (10 ms). As is the case for almost all speech features, delta and acceleration vectors ($\mathbf{\Delta, \Delta\Delta}$) were also found to be useful.

Interestingly if we used the $abs(\cdot)$ activation function at test time instead of the $log(1 + exp(\cdot))$ rectification (column **activation**) better results were achieved. This is even though most features have a matched feature with negative weights. It may be that the biases of the units prevent the features from being detected at small values; however, the sum of units is eventually transformed by $log(1 + abs(\cdot))$ so

pool-type	pool-window	$\Delta, \Delta\Delta$	activation	PER (dev)	PER (test)
$\log \sum(\cdot)$	400	Yes	abs(input)	21.2, 21.1, 20.9 (**21.1**)	22.7, 22.9, 22.8 (**22.8**)
		Yes	relu(input)	23.6, 23.9, 23.9 (**23.8**)	25.5, 25.8, 25.7 (**25.7**)
$\log \sum(\cdot)$	400	Yes	abs(input)	21.2, 21.1, 20.9 (**21.1**)	22.7, 22.9, 22.8 (**22.8**)
		No		21.6, 21.8, 21.8 (**21.7**)	23.5, 23.4, 23.8 (**23.6**)
$\sum(\cdot)$	400	Yes	abs(input)	23.3, 23.7, 23.8 (**23.6**)	29.7, 29.6, 30.3 (**29.9**)
$\log \sum(\cdot)$				21.2, 21.1, 20.9 (**21.1**)	22.7, 22.9, 22.8 (**22.8**)
$\sum \log(\cdot)$				22.0, 22.1, 22.1 (**22.1**)	23.5, 24.0, 23.7 (**23.7**)
$\log \sum(\cdot)$	400	Yes	abs(input)	21.2, 21.1, 20.9 (**21.1**)	22.7, 22.9, 22.8 (**22.8**)
	160	Yes	abs(input)	21.4, 21.5, 21.3 (**21.4**)	23.3, 23.5, 23.1 (**23.3**)

Table 3.2: Table showing phone error rate (PER) of a DNN HMM system with 2 layers trained on TIMIT, using ReLU features as inputs. Each row represents the result for a different combination of preprocessing options. The configuration $\log \sum(\cdot)$ with pool-window 400, using $\mathbf{\Delta}, \mathbf{\Delta\Delta}$ is repeatedly listed in lines 1, 6, 8, to make it easy to compare with other configurations that differ in only one setting. Each of these experiments was performed using hidden activities from a Gaussian-ReLU RBM with 80 input dimensions and 120 hidden units.

# of layers	dev	test
2	21.1, 20.8, 21.2 (**21.0**)	22.9, 23.0, 22.9 (**22.9**)
3	20.9, 20.6, 21.1 (**20.9**)	22.5, 22.2, 22.1 (**22.3**)
4	20.6, 20.5, 20.7 (**20.6**)	22.2, 22,1, 22.3 (**22.2**)
5	20.5, 20.6, 20.8 (**20.6**)	22.3, 22.1, 22.4 (**22.3**)
6	20.1, 20.7, 20.1 (**20.3**)	22.1, 22.0, 22.2 (**22.1**)
7	20.5, 20,6, 20.5 (**20.5**)	22.5, 22.1, 22.4 (**22.3**)

Table 3.3: PER of pretrained models of different depth using best configuration observed in table 3.2 - log(sum) pooling over 25 ms windows, striding with 10 ms, using $abs\,(\cdot)$ activation.

it is unlikely that small values contribute much to the final classifications. What is more likely is that approximate replication of the input features makes learning with SGD more difficult.

We used the configuration in table 3.2 that produced the lower PER on the validation set — $log\,(sum\,(\cdot))$ pooling of $abs\,(\cdot)$ activations over 25 ms of speech and striding by 10 ms — to train a Deep Belief Network (DBN) [Hinton et al., 2006]. The input to the DBN was 15 frames of features with delta and acceleration vectors. We trained a 7 layer model on TIMIT using the same configuration as the baselines in chapter 2 - 2000 sigmoid units in head layer. Table 3.3 shows a summary of the PER from triplicate experiments at depths 2-7. Pretraining does not seem to improve results for raw speech features - as it did for FBANK and MFCC inputs. In addition deeper models do no seem to produce much better results than shallower models.

The same configuration was applied to WSJ si-84 dataset. Table 3.4 shows the results on the validation set (test-dev03) and the test set (test-eval92). On the validation set, the features perform worse than MFCCs and FBANKs. But on the test set they perform slightly better than MFCCs but not as well as FBANKs.

features	run 1	run 2	run 3	average
MFCC	9.7 / 6.0	9.7 / 5.8	9.9 / 5.6	9.7 / 5.8
FBANKS	9.3 / 5.3	9.2 / 5.4	9.6 / 5.4	9.4 / 5.4
Gaussian - ReLU RBM	10.7 / 5.8	10.6 / 5.6	10.7 / 5.8	10.7/ 5.7

Table 3.4: WER of DNN-HMMs system on WSJ 14 hour subset (*si-84*). 120 units were used to model 5ms segments of raw speech signals

pool-type	pool-window	pool-stride	dim-reduction algorithm	PER
$\log \sum(\cdot)$	400	160	None (baseline)	22.8
			Autoencoder	23.2
			LDA	24.0
$\log \sum(\cdot)$	400	160	Autoencoder	23.2
$\sum \log(\cdot)$				22.8
$\sum(\cdot)$				22.6

Table 3.5: Table showing the test set PER of a DNN HMM system trained on TIMIT with 2 layers using a low dimensional representation of features. The inputs to the dimensionality reduction algorithms were the pooled activities of a Gaussian-ReLU RBM with 80 input dimensions and 120 hidden units. Note that the autoencoder features on the sum-pooled units were log transformed. None of these models were pretrained.

3.3.1 Dimensionality Reduction

As mentioned in section 3.2.2, there is a large redundancy in the features. In this section we explored the impact of dimensionality reduction of features on speech recognition accuracy. Table 3.5 shows the PER of a DNN HMM system with 2 layers, using a low-dimensional representation of features. Individual frames of the features computed using the best configuration in table 3.2 were reduced to 40 dimensions using an autoencoder. To do so an autoencoder with one hidden layer of 40 ReLU units was trained to reconstruct the input frames. The weights into the units were constrained to be positive so that only positive correlations between features were modelled. This procedure would capture the redundant features because we used the $abs\,(\cdot)$ activation function, which only cares about the angle of weight vectors, not the positive or negative direction. We compared with Linear Discriminant Analysis in which we used the phoneme targets associated with data vectors to find a 40 dimensional projection of the data. Delta and acceleration vectors were computed and appended to the 40 dimensional representation of data. As before, 15 such frames were concatenated and used to predict the phone label of central location. The trained neural network was then used as a part of a DNN-HMM for phone recognition. It can be seen that the dimensionality reduction does slightly improve performance. However the reduction was not drastic enough to warrant further investigation - it seems unlikely that this would lead to performance comparable to that achieved with FBANK inputs.

3.4 Conclusion

In this chapter we established that it is possible to discover features in raw speech automatically with Deep Learning methods that perform almost as well as hand engineered features. However generative modelling of raw signals themselves does not seem to be the best approach for speech recognition. Features for turbulent flow are by nature difficult to derive, unless some spectral representation is chosen. In addition it is much easier to inject domain knowledge into spectral features than it is into linear features on raw signals because spectral representations have spectro-temporal characteristics that allow models such as convolutional neural networks to be applied. The subsequent chapters thus focus more on modelling spectral representations of audio signals.

Chapter 4

Learning Features Using Transforming Autoencoders

The model we used in the last chapter did not use domain knowledge in the discovery of features. Domain knowledge has, however, been shown to improve accuracy of neural networks in Computer Vision on tasks such as object recognition [LeCun et al., 1998, Krizhevsky et al., 2012]. Prior knowledge, such as geometry, can be explicitly built into the architecture of these models by using a convolutional architecture. It can also be provided implicitly by augmenting the training data with copies of the data that have been transformed in ways that do not change the class information [LeCun et al., 1998, Krizhevsky et al., 2012, Ciresan et al., 2011, Simard et al., 2003]. For example, affine transformations of images do not usually alter the target labels and many different affine transformations of each training image can be used to vastly expand the size of the training set. We show how to augment FBANK data for training DNN-HMM speech recognizers better in chapter 6.

A new way of introducing domain knowledge into a neural network was introduced in the idea of the "transforming autoencoder" that makes it possible to train a group of neurons called a "capsule" to both recognize when a visual entity is present and to provide explicit information about the position, orientation and scale of the current instantiation of the visual entity[Hinton et al., 2011]. Using a transforming autoencoder, a layer of these capsules can be trained without having to specify what visual entity should be detected by each capsule and without having to specify the poses of the visual entities. Once the lowest layer of capsules has been learnt, the explicit representation of the poses of the visual entities should allow higher levels of neural networks to build more precise models for recognition.

In this chapter we show that a similar approach can be used for extracting acoustic events from either the waveform or from a spectrogram. Unlike the features typically produced by neural networks, a detected acoustic event will have an explicit representation of its exact time of occurrence and its exact amplitude, in addition to the probability that it is present. It was our expectation that these explicit instantiation parameters would make it easy to detect more complex acoustic events as relationships between more primitive events and result in more accurate recognition results. This chapter expands on our work in Jaitly and Hinton [2011b].

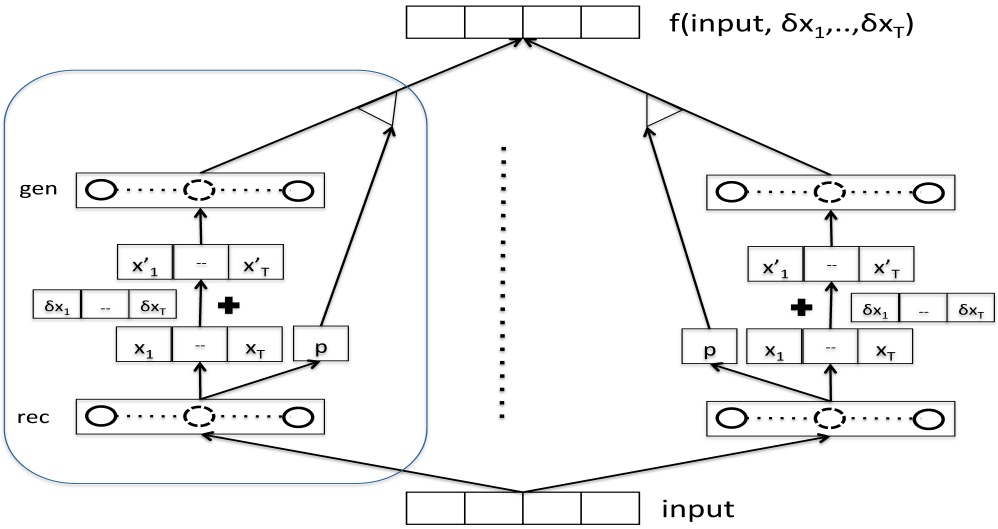


Figure 4.1: The architecture of a transforming autoencoder. The input is provided to a set of capsules, each of which uses a layer of logistic "recognition" units to compute a probability that its entity is present and to estimate the instantiation parameters that describe exactly how this instance of the entity is instantiated. The values of the instantiation parameters of each capsule are adjusted to take into account the global transformation that relates the input to the target output and the adjusted values are fed into a layer of logistic "generation" units which compute the contribution of the capsule to the output data. This contribution is then gated by the probability of activation of the capsule.

4.1 Transforming Autoencoders

Here we briefly describe the idea of transforming autoencoders as applied to images and we then describe its application to spectrograms.

A transforming autoencoder (see figure 4.1) receives both an input vector and a desired output vector that is related to the input vector by some simple global transformation such as a small translation of a whole image. It also receives an explicit representation of the global transformation. In the case of a pure translation this would be Δx and Δy, the translations in the x and y axes of the image. The bottleneck layer of the transforming autoencoder consists of the outputs of a number of capsules. In the case of a pure translation, each capsule would output the probability that its implicitly defined visual entity is present and the real-valued x and y coordinates of that entity relative to an origin that is implicitly defined by the capsule. The information about the global transformation is then injected by incrementing the x and y outputs of every capsule by Δx and Δy. The global transformations of the instantiation parameters were randomly generated for every data case, each time they were seen by the transforming autoencoder. The transforming autoencoder must then construct the transformed image from the transformed capsule outputs. All of the weights in a transforming autoencoder are learnt by back propagating the derivatives of the misfit between the actual and desired outputs. During the training phase, the different capsules learn to extract different entities or similar entities in different parts of the space of instantiation parameters in order to minimize the misfit between the final output and the target.

By explicitly injecting Δx and Δy, we encourage the transforming autoencoder to use the x and y outputs of each capsule to represent the real x and y coordinates of some visual entity. If it uses its x and y outputs in this way, it automatically produces the correct output image when the desired output image is held constant but the input image is translated by n pixels in the x direction and the Δx input is reduced by n.

One of the goals of a transforming autoencoder is to learn an "equivariant" representation, i.e., to learn a representation which changes in proportion to the transformation applied to the input. For example we would hope that moving images by Δx pixels in the x coordinate would change the associated instantiation parameters by Δx. Equivariant representations would allow us to build sharper higher level models because higher level representations would not lose resolution about the input - a property that invariant representations suffer from.

4.2 Application to Speech Signals

Here we describe how the method outlined above was applied to learn features for both waveforms and spectrograms. For this study we used the Arctic database [1] to train the autoencoder on waveforms and the TIMIT corpus [2] as the source of speech signals for the autoencoder on spectrograms. We used the Arctic database for the former experiment because it contains a large number (=1132) of utterances recorded for a single speaker and these utterances were recorded under strict recording conditions. This makes the task of learning features from waveforms more feasible. For the second task we used TIMIT database because we wanted to apply the features learnt to the task of phone recognition.

4.2.1 Waveforms

The entire set of waveforms for the Arctic database was first normalized so that the samples had a mean of 0 and a standard deviation of 1. The inputs were random segments of speech of duration 8 ms (=128 samples as the data was recorded at a sampling rate of 16kHz). The transforming encoder learnt to predict the central 4 ms (=64 samples) for each frame. The following transformations were applied to the data:

- **Translations:** The input speech signals were shifted left or right randomly by up to 32 samples. The value of the shift was added to the instantiation parameter corresponding to the temporal location of each capsule.
- **Amplitude Rescaling:** The intensities of the waveform samples were multiplied a random factor between .6 and 1.4. The instantiation parameters corresponding to intensity were multiplied by this scaling factor.

4.2.2 Spectrograms

Log absolute spectrograms were computed over 10 ms intervals with 5 ms stride between consecutive frames. 24 such consecutive frames (representing 10 + (24-1)*5 = 125 ms of waveform) were presented as input to the transforming autoencoder. The transforming autoencoder is required to estimate the the

[1] http://www.festvox.org/cmu_arctic/index.html
[2] http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC93S1

central 18 frames of the data after transformation (leaving 3 frames out on each side). The following three transformations were applied:

- **Translations:** The input frames of the spectrogram were shifted left or right randomly by up to three frames with equal probability.
- **Scaling:** The intensities of the spectrogram (before log transform) were scaled up or down by a random factor between .1 and 10. This was achieved by generating a random number between log(.1) and log(10) and adding it to the log spectrogram.
- **Frequency Re-sampling:** The log spectrograms were resampled in the frequency domain by a random rate between 0.8 and 1.2. The frequency resampling was achieved by first performing a cubic spline interpolation of each frame of the log spectrogram as a function of frequency, and then interpolating at integer multiples of the resampling rate.

4.3 A Way to Improve Transforming Autoencoders

In our preliminary experiments with spectrograms we observed that the transforming autoencoder was able to reconstruct transformed spectrograms by transforming instantiation parameters. However the predicted instantiation parameters themselves were not very equivariant - i.e. transforming an input in a certain way did not lead to an equal (or proportional) response in the code (see section 4.4.3). We devised a method that encouraged the encoder of the transforming autoencoder to learn equivariance by providing the transformed inputs as additional alternative inputs. The idea is that the instantiation parameters extracted from the original and the transformed inputs should differ by exactly the amount specified by the global transformation that relates the input and its transformed version.

The following regularization term was added for each type of transformation:

$$\sum_{i=1}^{N_c} p_i\left(\mathbf{v}\right)\left\{C_i\left(\mathbf{v}\right)+\partial c-C_i\left(\mathbf{g}\left(\mathbf{v},\partial c\right)\right)\right\}^2 \tag{4.1}$$

where N_c is the number of capsules, $\mathbf{g}\left(\mathbf{v},\partial c\right)$ is the result of applying transformation $\mathbf{g}\left(\cdot,\cdot\right)$ to vector $\mathbf{v}$ using parameter ∂c, $p_i\left(\mathbf{v}\right)$ and $C_i\left(\mathbf{v}\right)$ are the probability and the output respectively, of capsule i for data $\mathbf{v}$. This regularization term requires the capsule values to be equivariant to transformations of the data. i.e. if a transformation of ∂c was applied to the input data, the instantiation parameter computed by each capsule must also transform by ∂c if $p_i\left(\mathbf{v}\right)$ is significant. The regularization is used only to adjust the parameters related to the computation of the instantiation parameters, not to the computation of the capsule probabilities. Doing the latter resulted in much poorer models where equivariance of the generative features was strongly limited - this is possibly because a trivial way to reduce the value of equation 4.1 is to drive all the probabilities $p_i\left(\mathbf{v}\right)$ to 0.

4.4 Results and Discussion

4.4.1 Autoencoder on Raw Signals

The section deals with results obtained from a transforming autoencoder trained on raw speech signals. We trained an autoencoder with 100 capsules and 100 recognition and 100 generation sigmoid units per

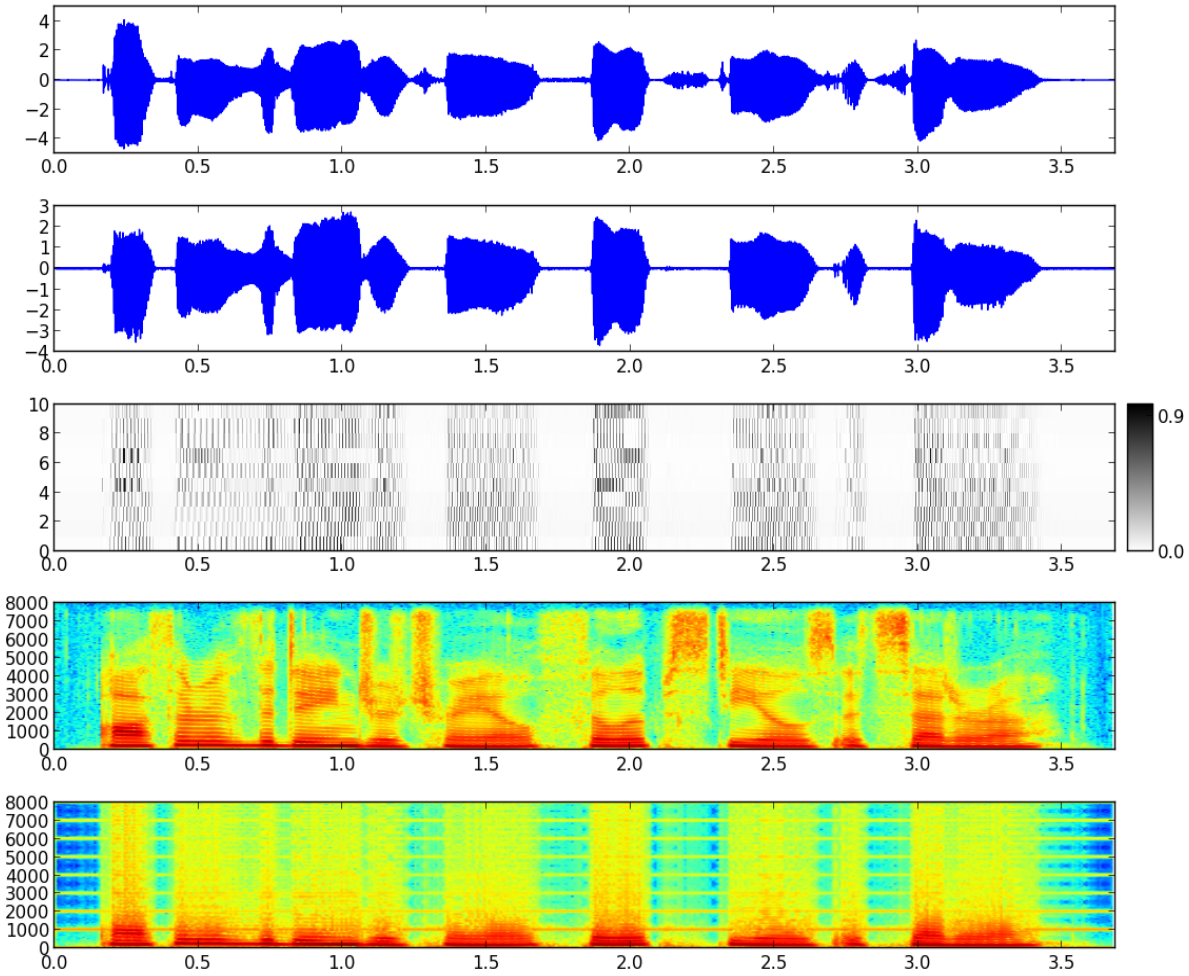


Figure 4.2: Reconstruction of raw signals with a transforming autoencoder. (a) Audio recording of utterance - 'Author of the danger trail, Philip Steels, etc.'. (b) Reconstruction of utterance from transforming autoencoder. The banding pattern is seen because the reconstruction was computed with a stride of 16 samples. (c) Activities of 10 most active capsules (darker means more active). (d) Spectrogram of original utterance. (e) Spectrogram of reconstruction.

capsule. A regularization weight of 2 on the term in equation 4.1 was used to train the model.

Figure 4.2 shows a segment of waveform, its reconstruction from the transforming autoencoder (without any transformations), the probability of activations of the ten most contributing capsules (in red), and the spectrograms of the original and reconstructed signals. It can be seen that the reconstruction is quite poor compared to what we achieved with ReLU Gaussian RBMs. Better reconstructions can be achieved if we reduce the range of transformations, but results in features that are less equivariant, in spite of the regularization term in equation 4.1. Figure 4.3 shows a zoomed in view of the reconstruction, and the contributions of the 5 capsules that contribute most strongly to these segments.

Figure 4.4 shows how the instantiation parameters change when 5000 randomly chosen raw speech segments are transformed using the transformations used in training (see section 4.2.1). It can be seen that the codes have learnt to be equivariant to X-shifts and intensity changes.

Figure 4.5 shows root mean squared (RMS) reconstruction error between transformed input and

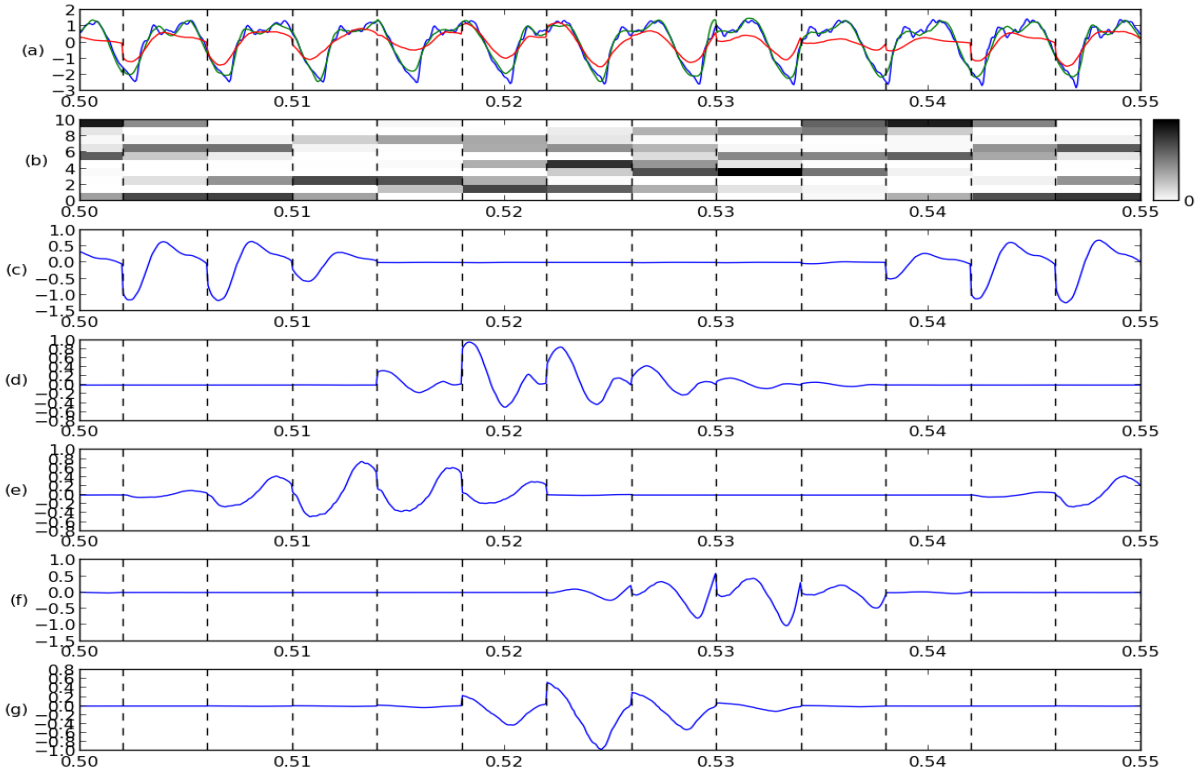


Figure 4.3: Zoomed in view of utterance in figure 4.2 between 0.5 to 0.55 seconds. This corresponds to the phoneme /er/ in word "author". (a) Original (in blue), reconstruction (in green) and sum of contributions from the 5 most contributing capsules. (b) Activities of 10 most contributing capsules (darker means more active). (c-g) Contributions of the 5 most contributing capsules (in sum of squared sense) in this segment. The most contributing capsules is in (c) and corresponds to capsule number 0 in (b). The least (in the top 5) contributing capsule is (g) and corresponds to capsule number 4 in (b). Individual frames are delimited by the dashed lines.

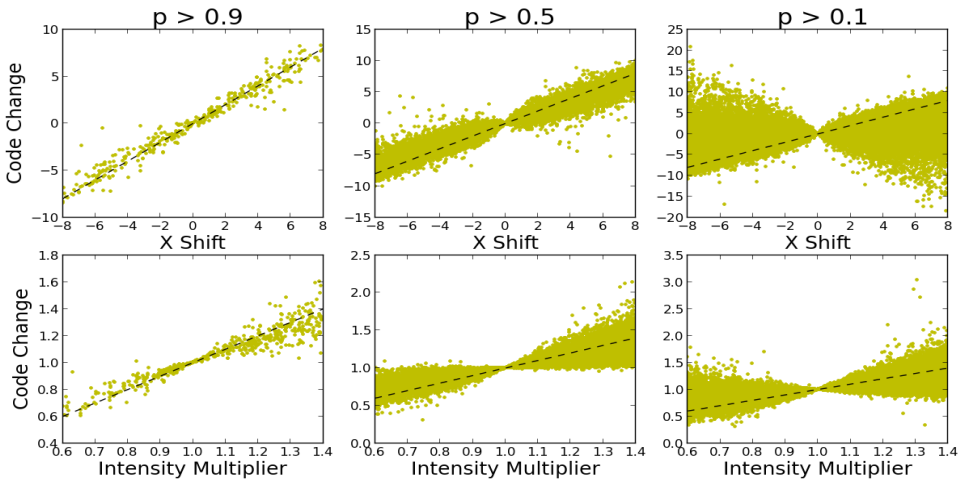


Figure 4.4: This figure shows the change in instantiation parameters as raw signals are transformed (section 4.2.1). *First row:* Plot of change in x-location instantiation parameter of capsules vs translation of raw signal within frame of reference. 5000 points were randomly translated by U(-8,8) samples. The change in x-location instantiation parameter of each capsule was recorded. This plot shows the translation on the x-axis, and the change in instantiation parameter on the y-axis. Only capsules that were more active than a given threshold in both the original and the transformed inputs are shown - the three columns use a threshold probability of 0.9, 0.5 and 0.1 respectively. *Second row:* Plot of relative change in intensity instantiation parameter of capsules vs relative intensity change of input. 5000 points were scaled linearly with a random number generated from U(0.6, 1.4). The relative change in intensity instantiation parameter of each capsule was recorded. This plot shows the scaling factor applied to inputs on the x-axis and the relative changed in intensity instantiation parameter on the y-axis. The same probability thresholds as above were used.

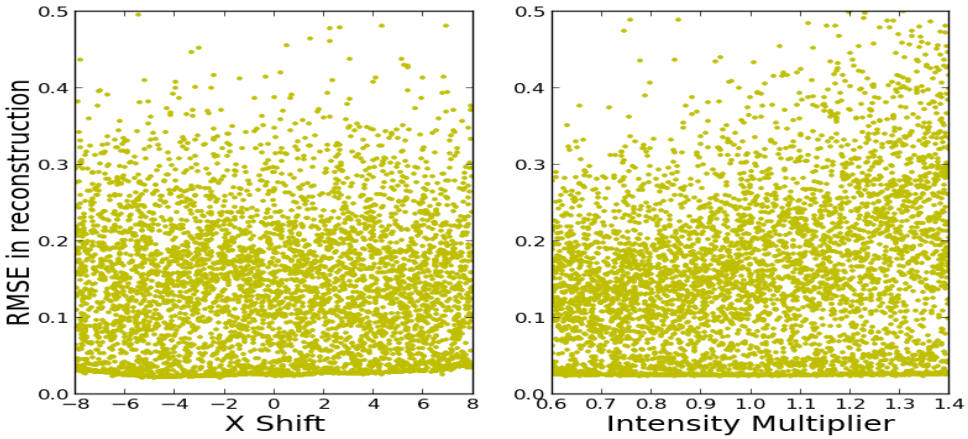


Figure 4.5: This figure shows the reconstruction errors as a function of change in instantiation parameters. The transformations in X-location and intensity are the same as described in figure 4.4. *Left Column* Mean squared error between transformed input and reconstruction generated by adding transformations to instantiation parameters computed on the original input. X-axis is the X-shift applied. Uniformly random jitters of U(-.5, .5) were applied to x-values show the points can be seen - the x-shifts themselves were discrete. *Right column* Mean squared error vs. intensity ratio. Clearly the transforming autoencoder has learned to reconstruct signals from instantiation parameters alone - changing instantiation parameters does not drastically influence the reconstruction error distribution.

transformed input generated by transforming instantiation parameters of original input. In general the autoencoder does a good job in reconstructing the input, but it can sometimes result in large reconstruction error. It could be that using ReLU recognition and generation units would lead to better results, given the relatively large orders of magnitude over which spectral features are spread out over.

4.4.2 Autoencoder on Spectrograms

We describe results obtained from a transforming autoencoder trained on spectrograms. As in the previous section, we had 100 capsules with 100 recognition and generation sigmoid units per capsule. A regularization weight of 5 on the term in equation 4.1 was used to train the model.

Figure 4.6 shows the spectrogram of an utterance, its reconstruction from a transforming autoencoder the probabilities of the 100 capsules and contributions of two different capsules[3]. It can be seen that the reconstruction is able to capture prominent aspects of the spectrum but misses out on the finer details. The capsules themselves show interesting contributions to the reconstructions. For example, the capsules in subplot (e) shows falling patterns associated with some formant transitions in the original spectrogram.

Figure 4.7 shows change in instantiation parameters that results when input is transformed. As before, it shows changes at three different probability thresholds. It can be seen that the transforming autoencoder has learnt to be relatively equivariant to intensity and frequency instantiation parameters. However, its response to changes in location of input seems to be much less equivariant. Spectrograms generally have strong correlations from one frame to another and the autoencoder may thus be ignoring

[3]The capsules were manually chosen by randomly scanning for a row (feature) in subplot (c) that seemed temporally correlated with a fricative pattern (high frequency energy bands) and a vocal pattern (formant pattern in the 2-3 kHz region) in the original spectrogram in subplot (a)

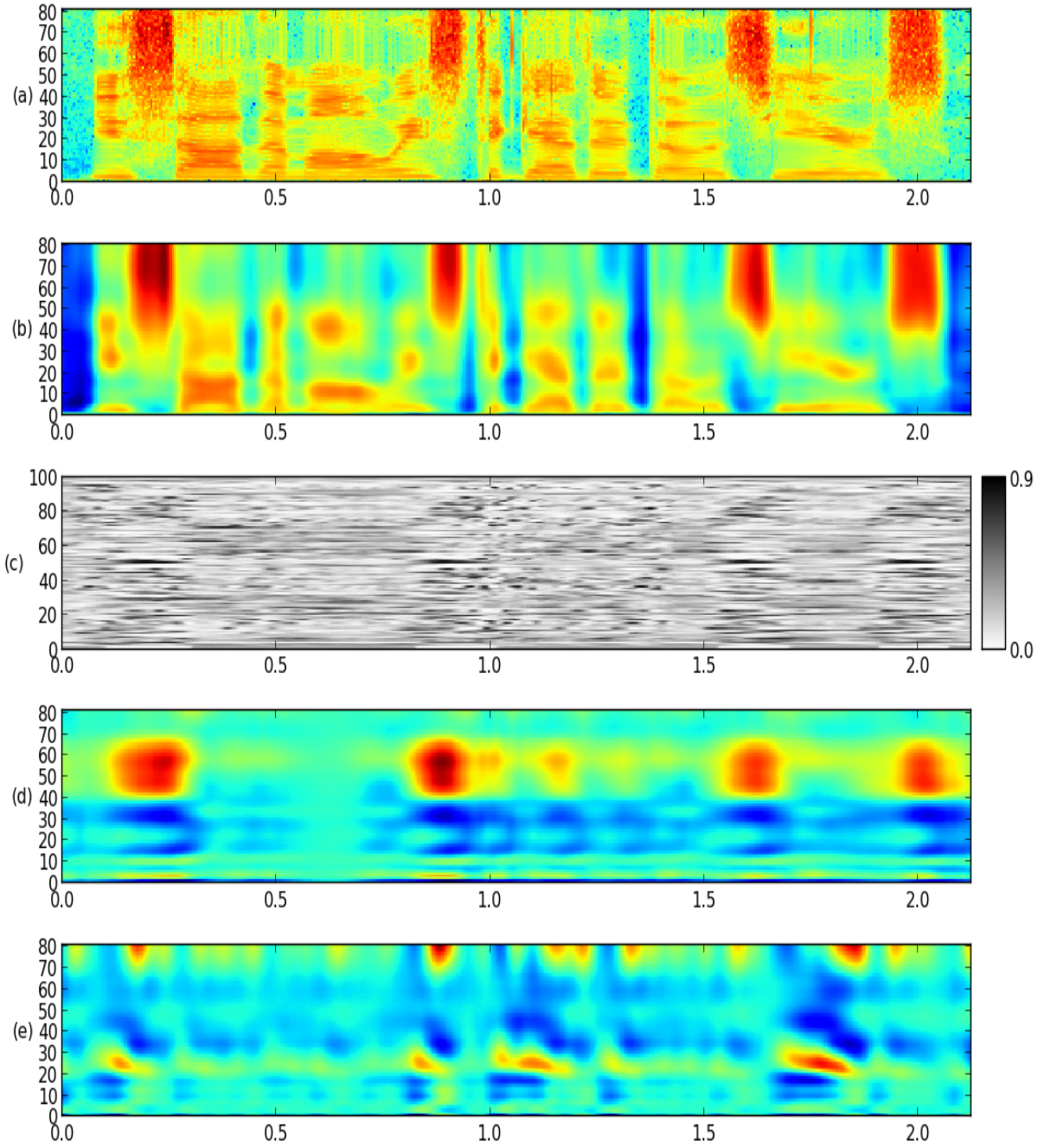


Figure 4.6: (a) Spectrogram for utterance - "You saw them always together those years". (b) Reconstruction of the utterance from capsules. (c) Probabilities for activations of capsules. (d,e) Contributions of two different capsules to the reconstruction. Each capsule is scaled differently. The capsule in (d) contributes energy to high frequency bands for fricative sounds, whereas the capsule in (e) contributes energy primarily to low-mid frequency bands and is possibly associated with one of the lower formants.

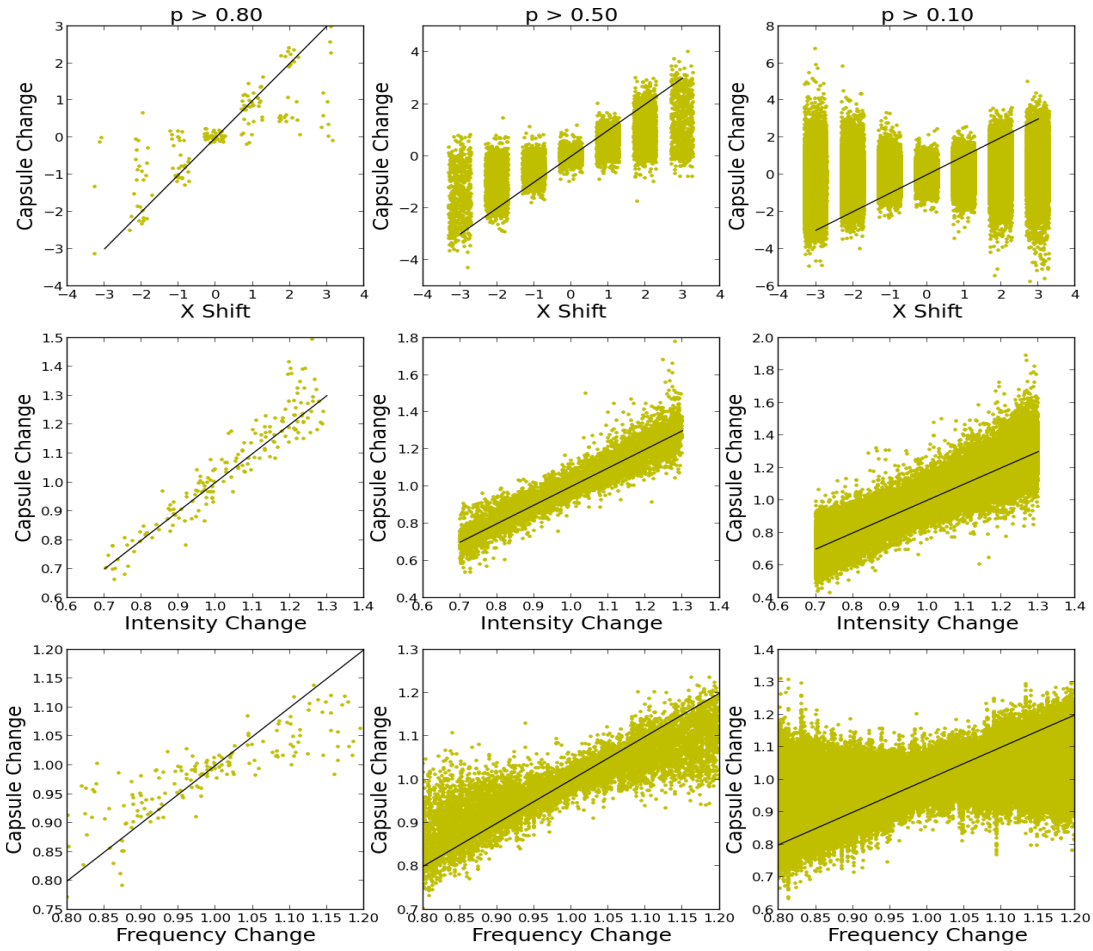


Figure 4.7: This figure shows change in instantiation parameters as the input is changed. Setup is analogous to that in figure 4.4. *First row:* Plot of change in x-location instantiation parameter of capsules vs. x-shift of input. *Second row:* Plot of change in intensity instantiation parameter of capsules vs. intensity change of input. *Third row:* Plot of change in frequency instantiation parameter of capsules vs. frequency rescaling of input.

temporal patterns, and instead focussing on getting the mean values correct. Also the way the input was transformed during training was discrete - frames were shifted discretely from -3 to +3 frames. It could be that this made learning invariance more difficult. Frequency rescaling, by contrast was done by generating a continuous variable and rescaling with spline fitting - and the network seems to behave better for this parameter.

Figure 4.8 shows root mean squared reconstruction error against the instantiation parameter deltas. As before, the error is between the transformed input and the reconstruction of the transformed capsules. It can be seen that the reconstruction error is relatively resilient to changes in instantiation parameters.

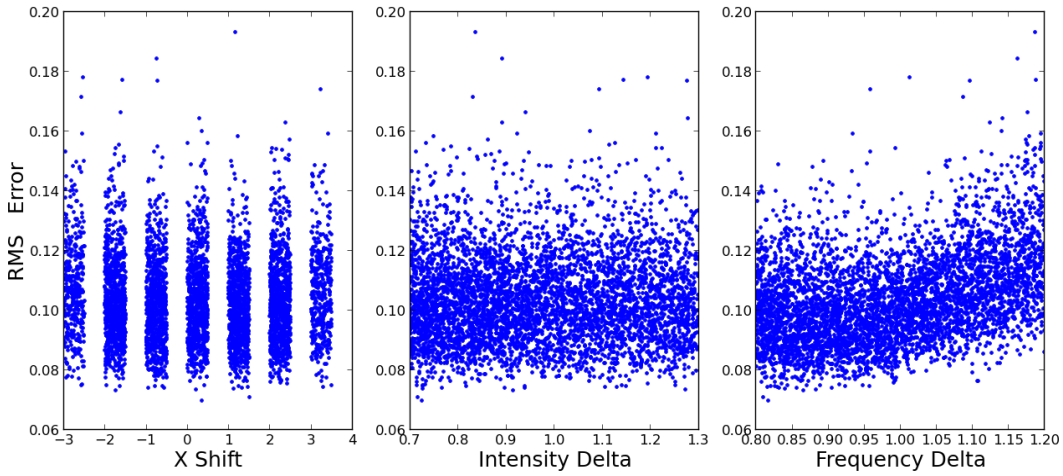


Figure 4.8: Figure shows RMS reconstruction error between a transformed spectrogram and its reconstruction generated from the transforming autoencoder by transforming the instantiation parameters of the capsules of the untransformed input. See section 4.2.1 for a description of the transformations.

# of units	# of capsules			
	50	100	150	200
50	0.086			
100	0.082	0.078	0.078	0.079
150	0.081			

Table 4.1: Reconstruction error of transforming autoencoder trained on spectrograms over different configurations (not all configurations were tried)

4.4.3 Impact of Regularization

Figure 4.7 shows change in instantiation parameters when the regularization in equation 4.1 was used. Figure 4.9 shows the same plot when the transforming autoencoder was trained without the regularization term in equation 4.1. As can be seen this model produces code that are less equivariant. The RMS error between transformed inputs and the reconstruction from transformed instantiation parameters however is lower (=0.072) for this model than typically achieved for the regularized models (see table 4.1). Thus the features have learnt to cooperate with each other in creating a reconstruction rather than acting as separate entities modeling structured parts of the input. This property is not desirable because the manifold over the instantiation parameters is likely to be more entangled making the instantiation parameters less useful for classification.

4.4.4 Peering Into the Mind of the Capsules

One of the most powerful aspects of capsules is the ability to inspect what they represent. This can be done by inspecting how the contribution of a capsule to the final output changes as we modify the values of its instantiation parameters. These 'fantasy' features represent the manifold over which different capsules contribute to output signals. Figure 4.10 shows how the output of one capsule trained on spectrograms changes as the frequency resampling and location instantiation parameters are changed over a grid of values. Also shown is how the output changes in response to the changing value of the

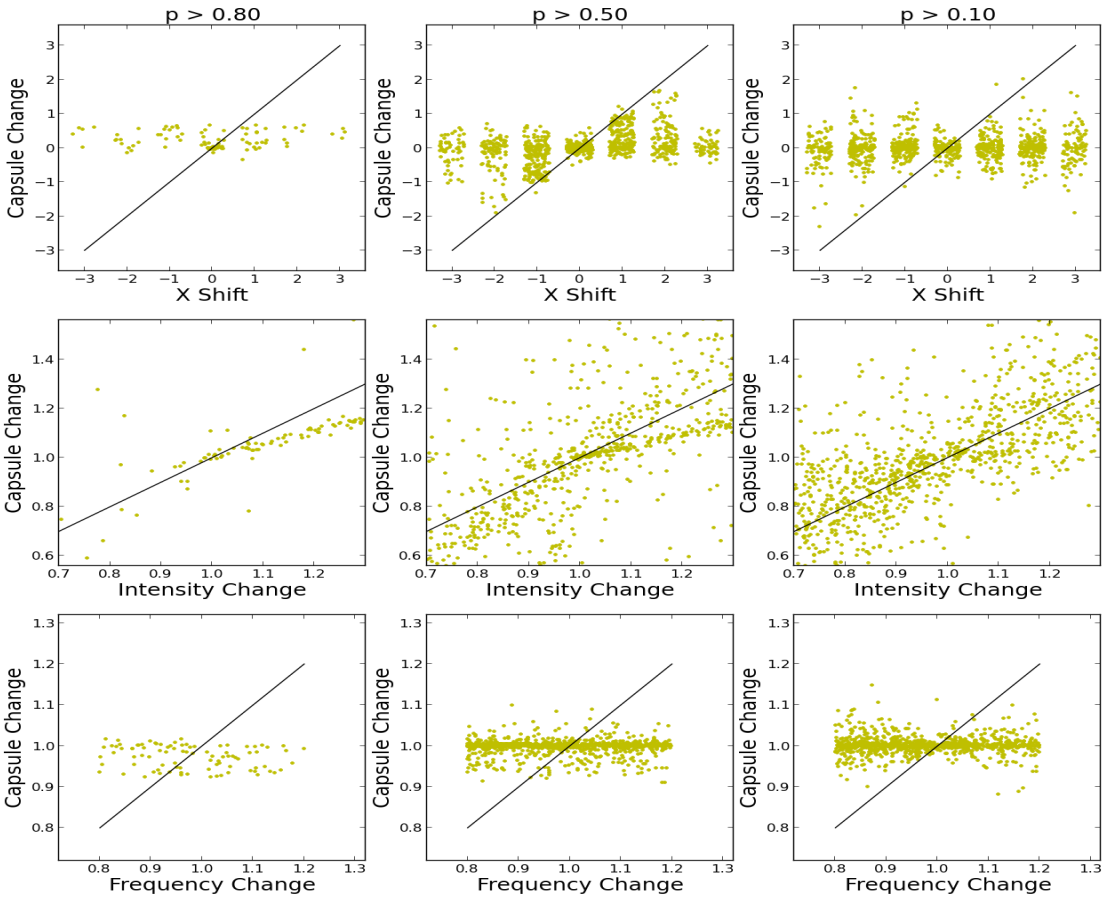


Figure 4.9: This figure shows change in instantiation parameters as the input is changed for a transforming autoencoder trained without the additional regularization in equation 4.1. The setup is analogous to that in figure 4.7 which used a transforming autoencoder that was trained with the regularization.

instantiation parameter corresponding to log(intensity). A similar trend was obtained for fantasies of capsules trained on waveforms.

4.4.5 Equivariance of Fantasy

Figure 4.11 shows how capsule fantasy changes in response to changes in instantiation parameters. In the ideal case, capsule fantasy would change exactly the same way as the instantiation parameters. However, it can be seen that this is not really the case for the instantiation parameters. In the next chapter we address this short-coming by using a hardwired decoder that respects the semantics of the instantiation parameters by design, instead of having to learn it.

4.4.6 Impact of Number of Units

In order to explore the effect of different parameters on reconstruction error, we trained a transforming autoencoder with different numbers of capsules and hidden units [4]. Table 4.1 shows a summary of these results for reconstruction error over the range of transformations described in section 4.2.2. It can be seen that reconstruction error is reduced by using a larger number of capsules and a larger number of hidden units.

4.4.7 Application to Phone Recognition

Features learnt by a transforming autoencoder on spectrograms in section 4.4.2 were used as inputs to a DNN-HMM system for phone recognition on TIMIT. A neural network with two layers of 2048 hidden units was used. A phoneme error rate (PER) of 22.7 was observed for the development set and 24.8 on the test set. These results were worse than the numbers reported for the ReLU-RBMs in the last chapter, and significantly worse than the numbers of filter bank inputs in section 2.3 where a network with similar architecture achieves a mean PER of 19.8% and 21.8% on the validation set and the test set respectively, with a standard error of 0.06% and 0.07% respectively.

Surprisingly, injecting knowledge into the autoencoder did not improve the accuracy of speech recognition. It is possible that a more appropriate use of the instantiation parameters of the capsules could result in a higher accuracy than our current results. It is also possible that small changes in the data result in large changes in the capsule codes producing an 'entanglement' of the code manifold which confounds neural networks trying to classify phoneme labels on the basis of the encoding. This situation would arise if the capsules learnt to cooperate with each other very strongly in reconstructing the data. Section 4.4.3 shows that without regularization capsules do indeed learn to cooperate with each other to reconstruct transformed outputs even without equivariance in the codes. With regularization the codes were more equivariant with respect to frequency and intensity instantiation parameters but the equivariance was still not perfect — figure 4.11 shows that the generative capsules are only partially equivariant. This is partly because it the decoder can learn to achieve equivariance in reconstruction without learning equivariant generative capsules.

In the next chapter we explore an alternative autoencoder which builds equivariance of reconstruction into the architecture of the autoencoder itself.

[4] For the sake of simplicity, the number of recognition units was fixed equal to the number of generation units in this study, although experiments indicate that more generation units result in better reconstructions

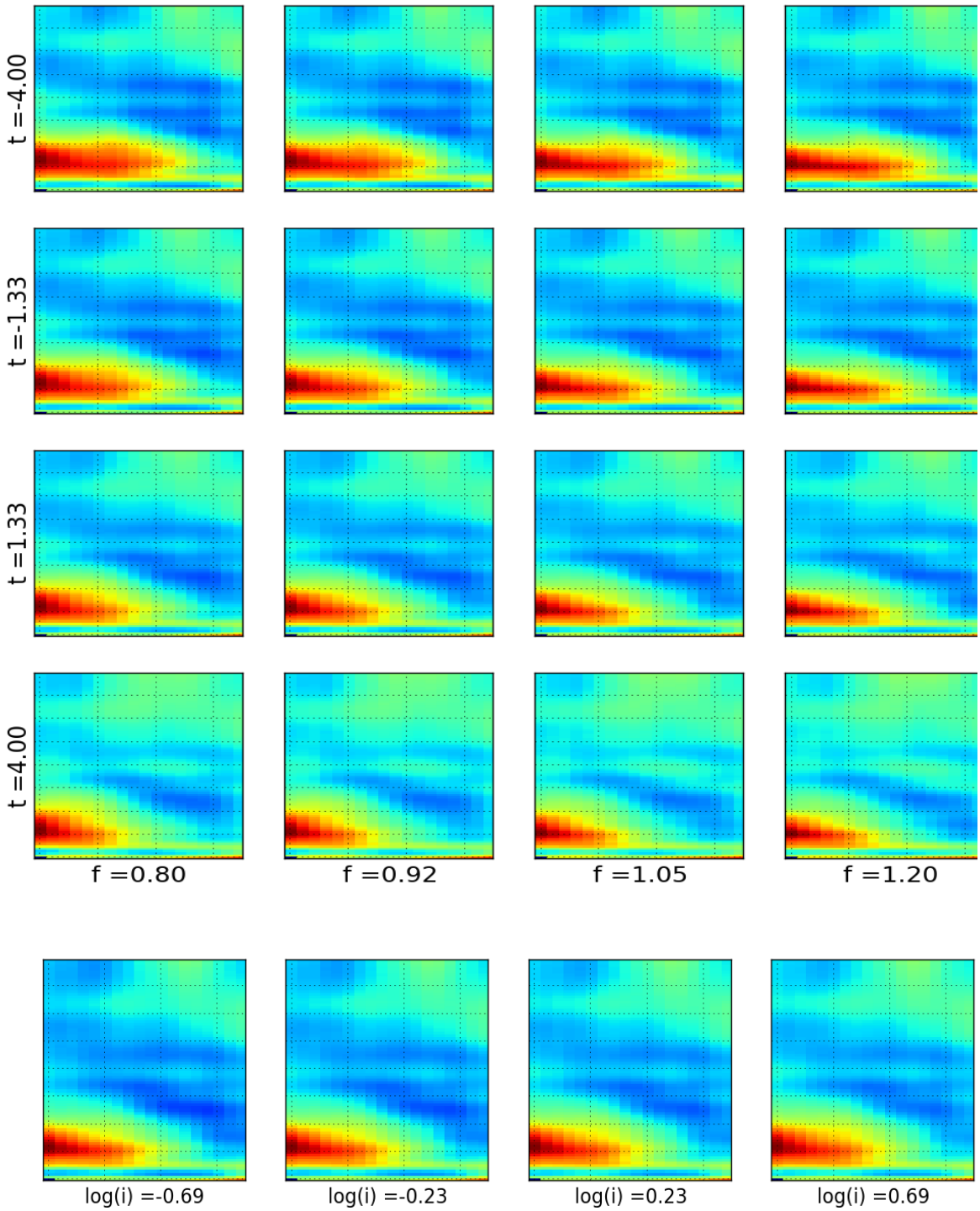


Figure 4.10: Capsule Fantasies. (a) Top 4x4 images: Along rows: Outputs of a capsule as instantiation parameter for frequency is set to 0.8, 0.92, 1.05 and 1.2 respectively. Down columns: variation in outputs of capsules as time variable of capsules is set to -4, -1.33, 1.33 and 4 respectively. (b) Bottom row: Outputs of a capsule as the instantiation parameter for intensity is set to log(.5), log(0.8), log(1.26) and log(2) respectively. Note: This figure should be viewed in color.

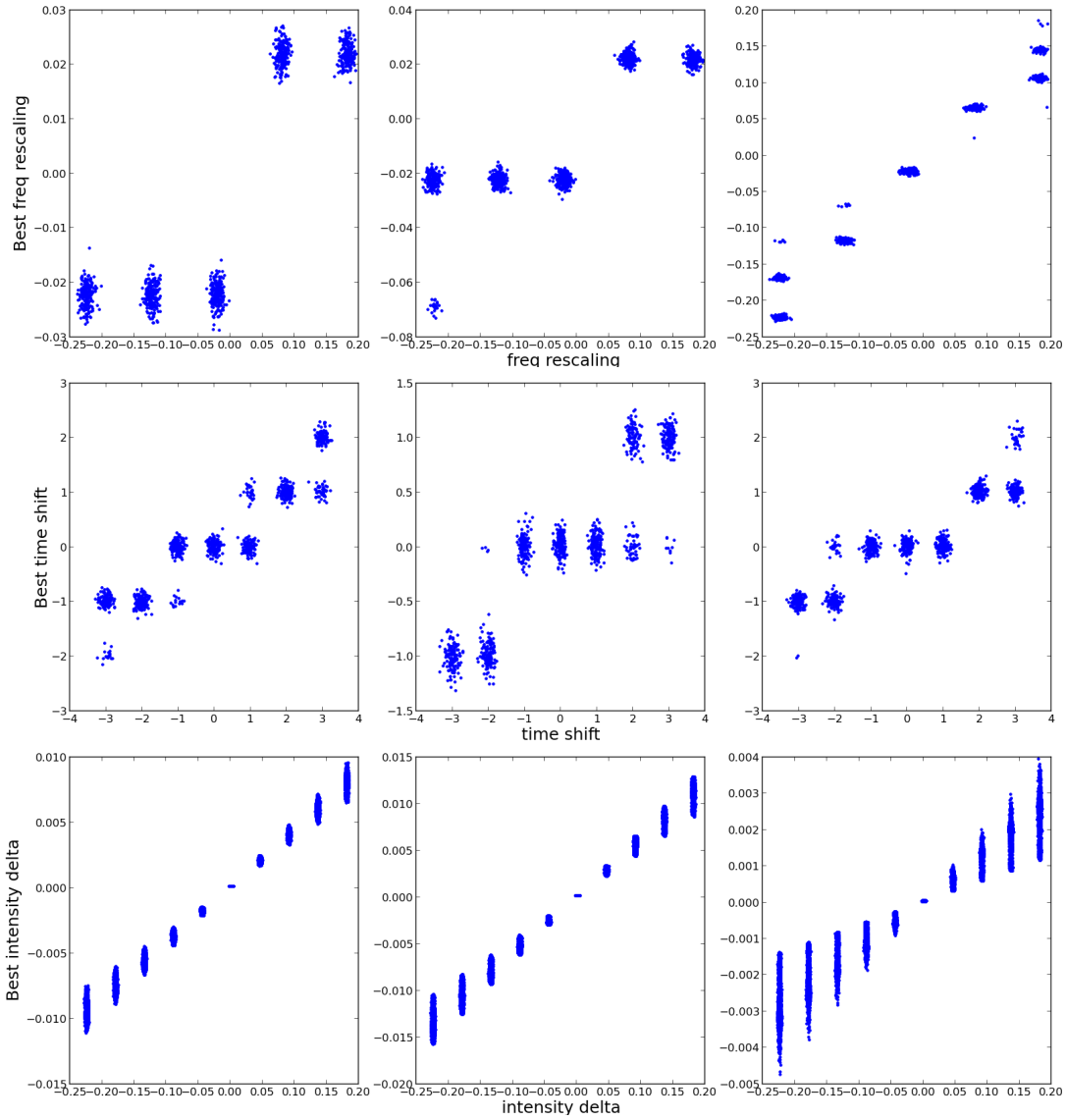


Figure 4.11: Change in fantasy with changes in instantiation parameters for three different capsules. (a) Frequency instantiation parameters of three capsules (along the columns) were changed and capsule fantasies were computed. The best frequency rescaling for each fantasy was computed by cross correlating reconstruction with frequency rescaled versions of the original contribution. Figure shows a plot of the change in instantiation parameter on the x-axis and the best frequency rescaling this corresponds to on the y-axis. It can be seen that changing instantiation parameters causes a similar change in the reconstruction. (b) Time instantiation parameters of three capsules (along the columns) were changed and capsule fantasies were computed. The best shift for each fantasy was computed by cross correlating reconstruction with shifted versions of the original contribution. Figure shows a plot of the change in instantiation parameter on the x-axis and the best shift that this corresponds to on the y-axis. It can be seen that equivariance is not perfect. Nevertheless changes instantiation parameter does results in corresponding change in fantasy. (c) Intensity instantiation parameters of three capsules (along the columns) were changed and capsule fantasies were computed. Mean intensity difference between fantasy after changing instantiation parameters and original fantasy was computed. Figure shows a plot of the change in intensity instantiation parameter on the x-axis and the best shift that this corresponds to on the y-axis. It can be seen that the capsule fantasies do not change in the same amount as the instantiation parameters. This is because many capsules add together to create the reconstructions. Thus the change in the fantasy of a single capsule is smaller in magnitude. Nevertheless, there is a strong linear relationship between change of instantiation parameter and the change in the fantasy.

4.5 Future Directions to Explore

We outline two shortcomings to capsules that merit future exploration in our opinion.

Shared Decoder

In our exploration the generative model for each capsule was independent of the other. Yet we would like them to have a very similar functional behavior that is guided primarily by the instantiation parameters. For example, when dealing with images and instantiation parameters of translation we would like each capsule to translate its output as a result of changing instantiation parameters. Such transformations need to be learnt by each capsule separately in the current model using a small number of generative units. This may be unreasonable when the domain is quite complicated. Instead a better approach may be to learn one generative function that is applied to the instantiation parameters of each capsule separately. This would have the added advantage that it would make it more difficult for capsules to cooperate during learning - any change to the generative model would effect all reconstructions.

Local Capsules With Weight Sharing Over Patches

In a transforming autoencoder a capsule can be active at only one location in the input, as specified by its instantiation parameters. This is limiting because patterns may be repeated at different locations in the input image and redundant capsules would have to be learnt. Convolutional neural networks avoid this problem by using filters convolutionally over patches of the image and pooling their responses over tiled patches. A transforming autoencoder could also avoid this limitation by using capsules that operate over local tiled patches (possibly with some overlap). Applied this way, the capsule instantiation parameters would preserve equivariance that is lost by convolutional neural networks while avoiding the computationally intensive convolutions at the same time.

Addressing the Limited Equivariance of Location

We observed that the instantiation parameter related to location was not very equivariant. This is possibly because spectral data is very smooth - the value at a frequency bin in one frame is almost identical to the value in the same frequency bin at the previous frame and the next frame. The transforming autoencoder can achieve a reasonably good reconstruction by simply predicting a vague value of location and placing mean values in the frequency bins around those locations. Removing temporal correlations with preprocessing such as PCA would remove this strong structure in the input, and possibly cause the models to be more sensitive to temporal locations of patterns.

Chapter 5

Learning Features Using Deformable Templates.

It was our hypothesis in the last chapter that different capsules may be cooperating with each other very strongly, producing codes that are hard to use for discrimination. One way in which this cooperation is learnt is through the neural network decoder - gradients received by the different capsules during training are backpropagations of the same vector - the difference between the target and the reconstruction at the last layer of the neural network - through the generative networks of the capsules.

In this chapter, we attempt to constrain the amount of co-adaptation possible between different capsules by creating a decoder that is not a neural network, but a function that respects the semantics of the domain - for example in images this function would be a graphics programs whose inputs are parts and their poses.

Here we first describe the idea using images of digits and faces as examples in section 5.1. We then present its application to frames of spectrograms and patches of FBANK spectra. Our description of the application of this idea to images is brief since the thesis is about speech, not vision; we present more details in the later sections which deal with speech. Note however, that the same code was applied to both domains. We refer the reader to Tieleman [2014] for a much more extensive application of this idea (and its extensions) to images. The application to frames of spectrograms has also been described in Jaitly and Hinton [2013a].

5.1 Autoencoder With Templated Parts

Figure 5.1 presents an overview of the autoencoder we use to learn capsules. The autoencoder reconstructs an input data point $\mathbf{v} \in \mathcal{D}$ as follows. The encoder estimates the instantiation parameters $\mathbf{z}_k$, $1 \leq k \leq K$ for each of the K parts and probabilities' p_k that the part exists in the data using its parameters, $\mathbf{\Theta}$ (neural network weights and biases). The decoder transforms the prototype $\boldsymbol{\theta}_k$ of each part, k, by a fixed transformation function (such as a 2D or 3D rotation), $f(\mathbf{z}_k; \boldsymbol{\theta}_k)$, using the instantiation parameters, $\mathbf{z}_k$ estimated for each part. The reconstruction $\mathbf{v}'$, is computed by combining the

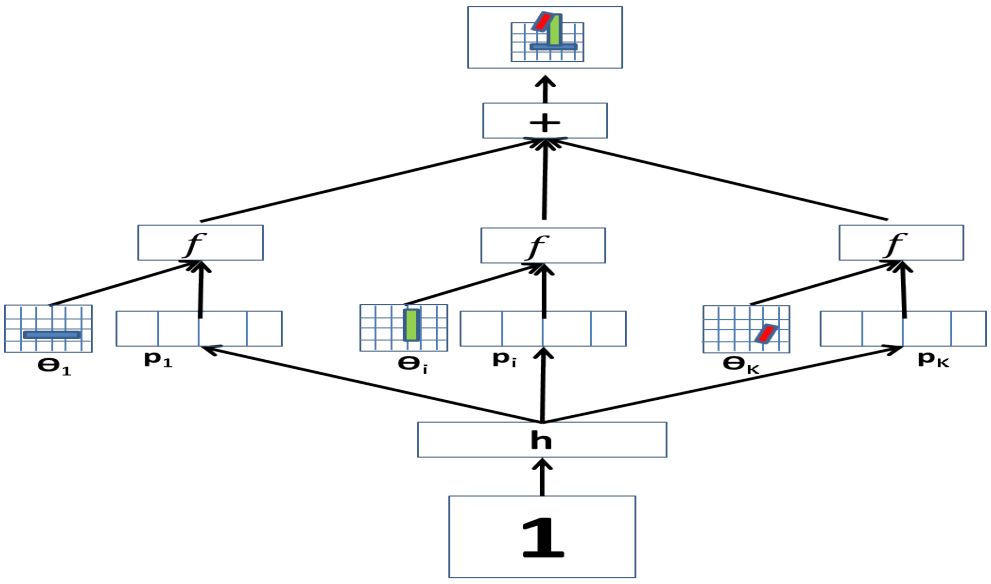


Figure 5.1: Overview of the method. An encoding neural network estimates instantiation parameters $(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_K)$ for K capsules. Each dimension of the instantiation parameters, $\mathbf{p}_i$, represents the value of a specific property of the feature (the dimensions of the capsules are selected as part of the design of the autoencoder, to capture important properties, for example poses for images). The decoder, computes $f(\mathbf{p}_i; \boldsymbol{\theta}_i)$ for each capsule, $1 \leq i \leq K$, using the capsule instantiation parameters $\mathbf{p}_i$, which depend on the data case, and parameters $\boldsymbol{\theta}_i$ which are specific to the capsule i. The reconstruction from the decoder is computed using $\mathbf{v} = \sum_i f(\mathbf{p}_i; \boldsymbol{\theta}_i)$.

transformed parts, using the probabilities π_k as weights:

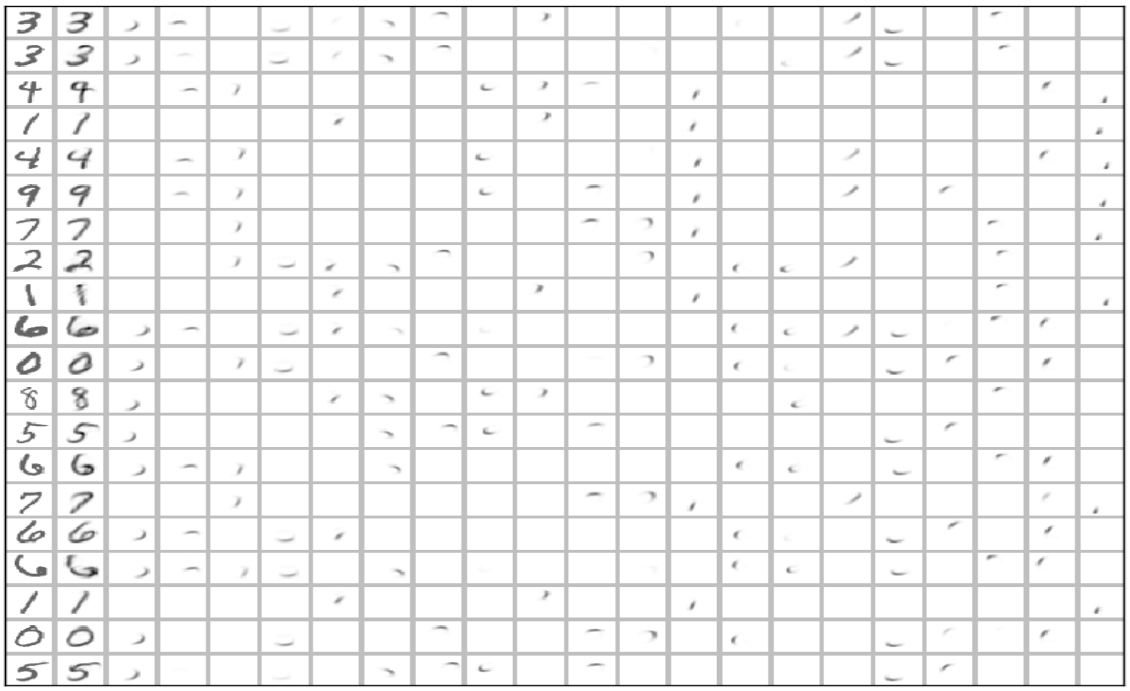
$$\mathbf{v}' = \sum_k \pi_k f(\mathbf{z}_k; \boldsymbol{\theta}_k) \tag{5.1}$$

Learning is done by finding parameters $\hat{\boldsymbol{\Theta}}$ for the neural network, and parameters $\hat{\boldsymbol{\theta}_1}, \cdots, \hat{\boldsymbol{\theta}_K}$ for the templates, that minimize the reconstruction error over the training data, i.e.:

$$\left(\hat{\boldsymbol{\theta}_1}, \cdots, \hat{\boldsymbol{\theta}_K}, \hat{\boldsymbol{\Theta}}\right) = \arg\min \boldsymbol{\theta}_1, \cdots, \boldsymbol{\theta}_K, \boldsymbol{\Theta} \sum_{\mathbf{v} \in \mathcal{D}} \|\mathbf{v}' - \mathbf{v}\|^2 \tag{5.2}$$

Because the decoding function, $f(\mathbf{z}; \boldsymbol{\theta})$ is chosen to be differentiable, its derivatives with respect to both the instantiation parameters, $\mathbf{z}$, $\frac{d}{d\mathbf{z}} f(\mathbf{z}; \boldsymbol{\theta})$, and its parameters, $\boldsymbol{\theta}$, $\frac{d}{d\boldsymbol{\theta}} f(\mathbf{z}; \boldsymbol{\theta})$ can be computed. Correspondingly the derivatives of the reconstruction error can also be computed with respect to each of $\boldsymbol{\theta}_k$, and $\mathbf{z}_k$. The latter of these can be passed on to the neural network and back-propagation can be used to compute the derivatives of the error with respect to the parameters, $\boldsymbol{\Theta}$ of the encoding neural network. Using all these derivatives, we can perform learning in the model using any of the standard derivative-based optimization schemes. Figure 5.2 shows the application of this autoencoder to images of digits from the MNIST database [LeCun and Cortes, 1998]. In this application the encoder predicts instantiation parameters and probabilities for 20 templates. The instantiation parameters are the location (x_i, y_i), $1 \leq i \leq 20$, of the center of the templates in the reconstruction. The decoder transforms template i so that it is centered at (x_i, y_i) and adds together all the transformed templates.

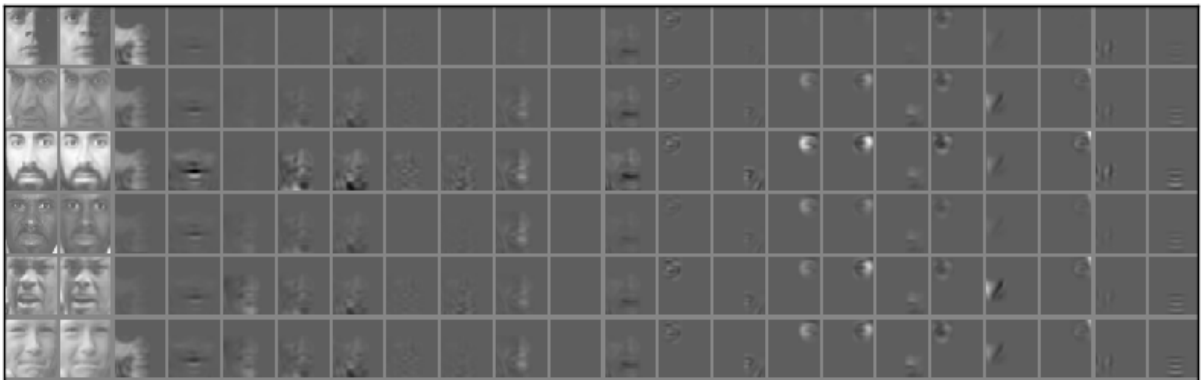
(a) Templates



(b) Autoencoder Reconstruction

Figure 5.2: (a) Templates for the 20 parts learnt by the capsules-autoencoder on MNIST images. (b) Contributions of templates to reconstructions of digits. The first column shows the original digits. The second column shows the reconstruction. The remaining columns display the contribution from the first/second/third/etc capsule.

(a) Templates for faces



(b) Autoencoder Reconstruction

Figure 5.3: (a) 20 of the 50 template parts learnt by the capsules-autoencoder on TFD. (b) Contributions of templates to reconstructions of faces. See 5.2 for description of the format of these figures.

The contribution of each template is weighed by the probability π_i that it is active. The subplot at the top shows the 20 templates that are learnt. The bottom plot shows how they are used by the autoencoder to reconstruct 20 example digits.

Figure 5.3 shows the application of this autoencoder to images of faces from the Toronto Faces Database [Susskind et al., 2010]. In this application the encoder predicts instantiation parameters and probabilities for 50 templates. The instantiation parameters are the location (x_i, y_i), $1 \leq i \leq 20$ of the center of the templates in the reconstruction, its intensity a_i and probability that it is on π_i. As with digits, the decoder transforms template i so that it is centered at (x_i, y_i) and adds together all the transformed templates. The contribution of each template is weighed by the product of its probability and its intensity $\pi_i a_i$. The subplot at the top shows a random subset of the 50 templates that are learnt. The bottom plot shows their contributions to the reconstruction of six randomly chosen examples.

5.2 Application to Frames of Spectrograms

5.2.1 Motivation

Spectrograms are a very powerful representation for encapsulating information in raw speech signals. However acoustic information is often spread over multiple frequency bins. For example a formant appears as increased energy over several frequency bins and not as a single, resolved peak. Discovering a formant peak is further complicated by the presence of other peaks such as other formants and the harmonic stack of the fundamental frequency (F0). Further, small changes in acoustics, such as a change in the frequency of a formant peak or F0 can cause a highly non-linear change in the spectrogram representation as the peaks hop from one frequency bin to the neighboring bins.

Mel log filter banks are a lower resolution version of spectrograms in which rescaled intensity infor-

mation from multiple neighboring bins are added together. The spacing between these bands is inspired by perceptual studies [Stevens et al., 1937, Davis and Mermelstein, 1980]. A side effect of the lower resolution of the filter-banks is that this problem of 'dimension hopping' is mitigated. The information about the precise location of the formant peak is obviously lost but robustness for speech recognition is gained because speaker to speaker variability is reduced by the wide triangular windowing functions used to combine energy from neighboring frequency bins. Vocal Tract Length Normalization (VTLN), similarly, mitigates this problem by rescaling the frequency axis of spectrograms across speakers before the computation of Mel log filter banks. The transformation makes formant peaks more consistent from speaker to speaker [Lee and Rose, 1998].

We motivate the use of a template based autoencoder to discover spectral features through the following simple modification to how a Mel filter bank spectrum is computed. Traditionally, Mel filter-banks have fixed center frequencies and window widths, and windowing functions (e.g., triangular). Imagine a method which customizes how a Mel filter bank spectrum is computed by predicting the center frequency, the window width and windowing type. Such a method has the potential of being more sensitive than Mel filter-banks by integrating information across neighboring bins in a more precise manner depending on the data.

Here we use a less constrained approach than that suggested above. We use prototypical patterns or *templates* that are frames of spectrograms. The templates are stretched or compressed along the frequency axis, and are blended together at different intensity levels to compute a reconstruction. The intensity level and the degree of stretching / compression required for each template to make a good overall reconstruction is predicted by using an encoding neural network. The parameters of the neural network and the templates are learnt by minimizing the reconstruction error over training cases.

We used a decoder that was appropriate for the domain - it takes prototypes and stretches or compresses their frequency axis in a manner akin to VTLN and computes a reconstruction. The degree of stretching / compression is specified by the instantiation parameters computed by the encoder. In addition to computing the reconstruction the decoder also computes the gradients of the reconstruction error with respect to the prototypical patterns and the predicted transformations. These gradients are backpropagated through the neural network encoder during learning and neural network and template parameters are learnt by stochastic gradient descent with momentum using an ADA-grad like rescaling of the gradient of each parameter by a running estimate of the standard deviation of the gradient [Duchi et al., 2011].

We applied this method to learn a model for frames of spectrograms from the TIMIT and WSJ databases ([Garofolo et al., 1993, Paul and Baker, 1992]) and used this model to augment features used in speech recognition.

5.2.2 Autoencoder Details

Figure 5.4 shows an overview of the autoencoder. A neural network (labeled E) takes as input a frame of a spectrogram, $\mathbf{v}$ and predicts T sets of values $\mathbf{z}_t = (f_t, a_t)$, $1 \leq t \leq T$. Each set, (f_t, a_t), specifies the parameters of the transformation (described below) that is applied to the corresponding template frame $\mathbf{s}_t$. The decoder applies these transformations to the templates, and adds the resulting vectors to create the reconstruction of the input frame. The model is learnt by minimizing the total reconstruction error over the database.

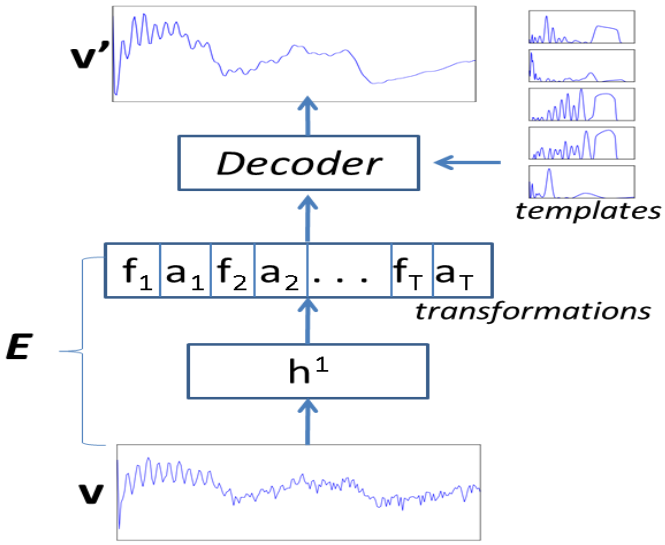


Figure 5.4: Overview of the autoencoder. The encoding neural network predicts transformation variables for the template patterns. The decoder applies the transformations to the template patterns and computes the reconstruction, $\mathbf{v}'$, of the input, $\mathbf{v}$, from the transformed templates.

Model Details

Specifically the autoencoder takes an input data frame $\mathbf{v}$, and computes a reconstruction $\mathbf{v}'$ as follows. The input data frame $\mathbf{v}$ is forward-propagated through the neural network using its parameters $\mathbf{\Theta}$ (weights and biases of the layers), to compute the encoding $\mathbf{h} = [f_1, a_1, f_2, a_2, \cdots f_T, a_T]' = \mathcal{N}(\mathbf{v}, \mathbf{\Theta})$, where $\mathcal{N}$ represents the neural network function (not to be confused with normal distributions). The decoder transforms each template $\mathbf{s}_t$ as follows. It first resamples the periodogram that the template represents at a new rate f_t, using linear interpolation. For example, if $\mathbf{s}_t = [s_1, s_2, s_3 \cdots]'$ and $f_t = 1.2$ then $\mathbf{s}'_t = [s1, (0.8s_2 + 0.2s_3), (0.6s_3 + 0.4s_4) \cdots]'$. Let $\mathbf{r}(.,.)$ represent this multivariate transformation function. Note that $\mathbf{r}(.,.)$ retains the property that consecutive peaks of harmonic stacks are equally spaced. The decoder multiplies $\mathbf{s}'_t$ by a_t and adds all the transformed templates together to compute a reconstruction for the input. Mathematically,

$$\mathbf{v}' = \sum_t a_t \mathbf{r}\left(\mathbf{s}_t, f_t\right) \tag{5.3}$$

Parameter Learning

Learning is done by finding parameters $\mathbf{\Theta}$ for the neural network, and $\hat{\mathbf{s}_1}, \cdots, \hat{\mathbf{s}_T}$ for the T templates that minimize the total reconstruction error, E, over the training data, i.e.:

$$\left(\hat{\mathbf{s}_1}, \cdots, \hat{\mathbf{s}_T}, \hat{\mathbf{\Theta}}\right) = \arg\min \mathbf{s}_1, \cdots, \mathbf{s}_T, \mathbf{\Theta} \sum_{\mathbf{v} \in \mathcal{D}} \|\mathbf{v}' - \mathbf{v}\|^2 \tag{5.4}$$

The above function is differentiable with respect to all the parameters $\mathbf{s}_1, \cdots \mathbf{s}_T$ as the bilinear interpolation operation is a simple linear operation. Similarly, the gradients of the error with respect to f_t and

a_t are also easily computed [1]. Gradients of the reconstruction error with respect to the neural network parameters are computed by back-propagating the gradients with respect to f_t, a_t, through the neural network.

Using all these derivatives, we can perform learning in the model using any of the standard derivative-based optimization schemes. For this chapter the parameters of the models were learned by performing stochastic gradient descent on the error function. The gradient of each parameter was rescaled by a running estimate of the standard deviation of the gradient. A small learning rate of 10^{-5} per average data case was used. The parameters of each template were constrained to have an Euclidean norm of 1, after each gradient update, by re-normalizing its length. This allows intensity values for different templates to be somewhat comparable.

In practice, the error function in equation 5.4 is modified by adding an L1 penalty to the template intensities a_t, i.e.

$$E = \sum_{\mathbf{v} \in \mathcal{D}} \|\mathbf{v}' - \mathbf{v}\|^2 + \lambda \|\mathbf{a}(\mathbf{v})\| \tag{5.5}$$

where $\mathbf{a}(\mathbf{v})$ is the vector of intensity instantiation parameters predicted for $\mathbf{v}$, i.e. $\mathbf{a}(\mathbf{v}) = [a_1, a_2, a_3 \cdots]'$, and a_t is the intensity of template t, predicted by the encoder, when data vector $\mathbf{v}$ is presented as input.

This penalty term attempts to make the solution more sparse, and produces more identifiable features.

5.2.3 Experimental Methods and Results

Spectrograms were exported from the Kaldi recipes described in chapter 2. The spectrograms were computed over 25ms intervals with a stride of 10 ms. Since the audio data was collected at a sampling rate of 16kHz, the frames were 201 dimensional ($16000 * 25 * 10^{-3}/2 + 1$). The spectrograms were used for unsupervised learning with the autoencoder. The discovered features were appended to Mel log filter banks features and used in a Hybrid Deep Neural Network Hidden Markov Model (DNN-HMM) for automated speech recognition.

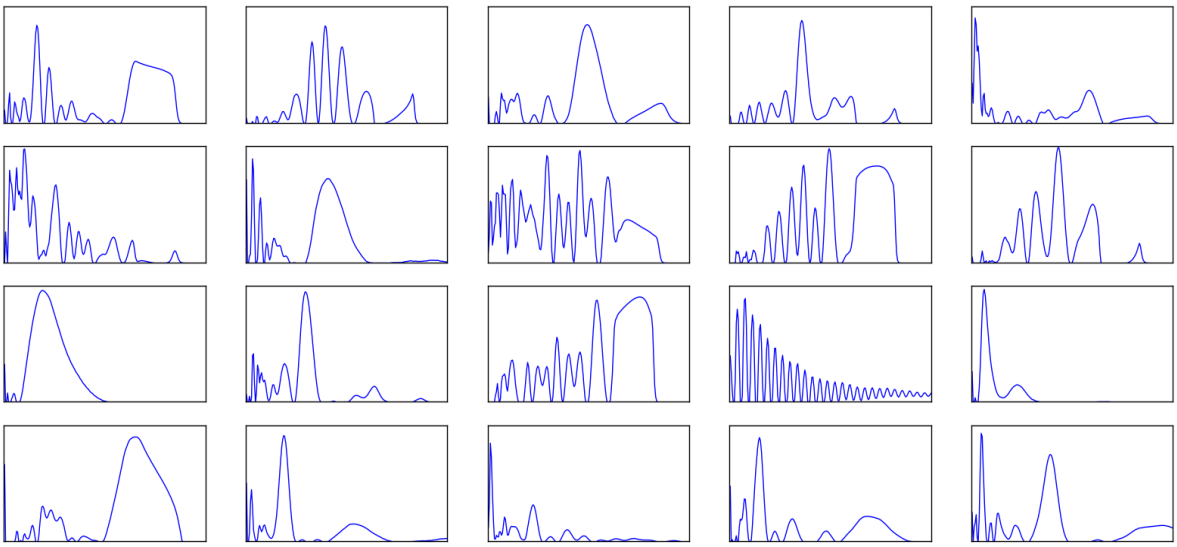
Feature Learning

An autoencoder model with templates was trained to reconstruct the spectrograms as described below (also see figure 5.4).

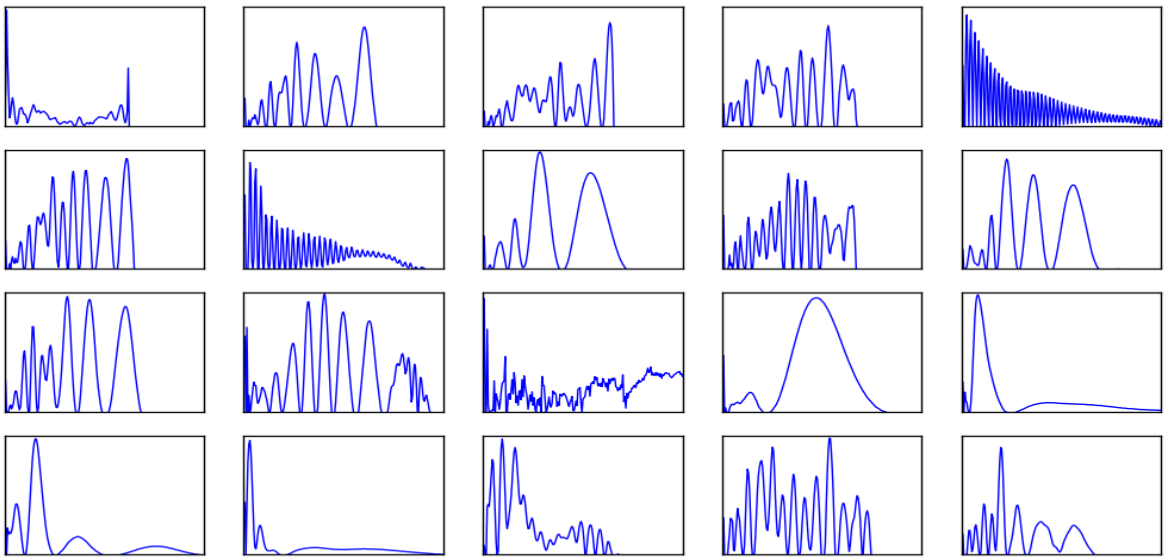
We used a two layer neural network with 2000 units in the first hidden layer, and $2T$ units in the second layer. The $2T$ units in the second layer correspond to T pairs of (f_t, a_t) values, $1 \leq t \leq T$. The f_t values were outputs of sigmoid units whose output range of 0-1 was rescaled to $-\log F, \log F$, (we used $F = \exp(.5)$). The a_t values were outputs of rectified linear units (ReLU) [Nair and Hinton, 2010]. Each template $\mathbf{s}_t$ frame was resampled at a new rate of $\exp(f_t)$ by taking interpolated values at $(0, \exp(f_t), 2\exp(f_t), ..)$, using bilinear interpolation. Negative values of f_t lead to stretching out of the templates (corresponding to increasing the frequencies of patterns) and positive values lead to compressing of the templates (corresponding to reducing the frequencies of patterns). The stretched or compressed template vector is then multiplied by a_t.

Figure 5.5a shows the templates learnt using the TIMIT database and an autoencoder with $T = 20$ templates. Figure 5.6 shows reconstructions of a spectrogram using these templates (each frame

[1] $\mathbf{r}(\mathbf{s}_t, f_t)$ is technically not differentiable w.r.t. f_t at $f_t = 1$ since the left and right derivatives may not be equal. However this is trivial and can easily be avoid by using higher order interpolations e.g., splines. For convenience we just used a value of 0 for this case.



(a) *TIMIT.*



(b) *WSJ.*

Figure 5.5: Template features discovered by our method on (a) TIMIT (b) WSJ. x-axis is frequency bin (0-200) and y-axis is intensity.

was separately reconstructed). It can be seen from figure 5.6 that a very good reconstruction of the spectrogram is possible from just 20 templates. Several templates have identifiable characteristics; for example templates 3 and 16 seem to correspond to fricatives as can been seen from their high intensity during fricative portions of the spectrogram; template 14 (column 4 from left, row 3 from top) in figure 5.5a clearly corresponds to a harmonic stack. The last panel of the figure shows reconstruction that is achieved by using 20 radial basis functions spanning the frequency axis. Clearly, the quality of reconstruction is much lower than that achieved from the templates.

Automatic Speech Recognition

For the speech recognition experiments, we used our features in a hybrid DNN-HMM system as described in chapter 2 [Bourlard and Morgan, 1993, Hinton et al., 2012a]. Neural networks were trained to predict the phoneme state labels, and the predictions were converted to scores that reflected the generative probability of the frame from the HMM model. These scores were fed into Kaldi, which performed the decoding. A fixed acoustic model scale of 1.0 was used in the DNN-HMM system. Before the neural networks were trained, we pretrained the weights using a Deep Belief Network (DBN) [Hinton et al., 2006].

Template intensities (but not frequencies) were appended to the 120 dimensional Mel log filter bank vectors. The Mel log filter bank dimensions were normalized to mean 0 and unit standard deviation. However, the template intensities were outputs of ReLU's, and we didn't think mean centering and normalizing would be an appropriate operation for these inputs. This is because a small template intensity, could possibly end up having a large negative value, for ReLUs. Instead the intensity values were rescaled by their standard deviation without mean centering.

A neural network was trained to predict the phoneme state label of a frame using the frame and +/- 7 frames of context. Learning was done using stochastic gradient descent on mini batches of size 100, with momentum of 0.0 and learning rate of 0.1 times the average gradient per case, for the first epoch. After that a momentum of 0.9 was used. At the end of each epoch the frame error rate (FER) was computed on a validation set. If the FER increased over the previous epoch, the learning rate was decreased by a factor of half, and the parameters were reset to the values at the start of the epoch. This process was repeated until the learning rate was decreased eight times. The test set was then decoded with the final parameters.

Table 5.1 shows the impact of sparsity penalty λ (in equation 5.5) and the number of templates used on the phone recognition accuracy. For these experiments we used two hidden layers of 2000 sigmoid units each. The networks were not pretrained. The sparsity penalty seems to have little impact on the PER. Using more templates does not improve PER either. Manually inspecting the templates learnt we found that when we used more templates, the neural network ignored some of them by predicting template intensities as small values, effectively creating dead templates. This is probably because a small number of templates is powerful enough to reconstruct the spectrogram (as can be seen from figure 5.6).

DBN pretraining was then performed by training Restricted Boltzmann Machines (RBMs) for 50 epochs for each layer. The bottom layer was trained as a Gaussian-Binary RBM while the others were trained as Binary-Binary RBMs. Unlike Mohamed et al. [2012] we used only 50 epochs of pretraining - further pretraining did not seem to help the PER after decoding for our system. The DBN parameters were used to seed the DNN-HMM system discussed next.

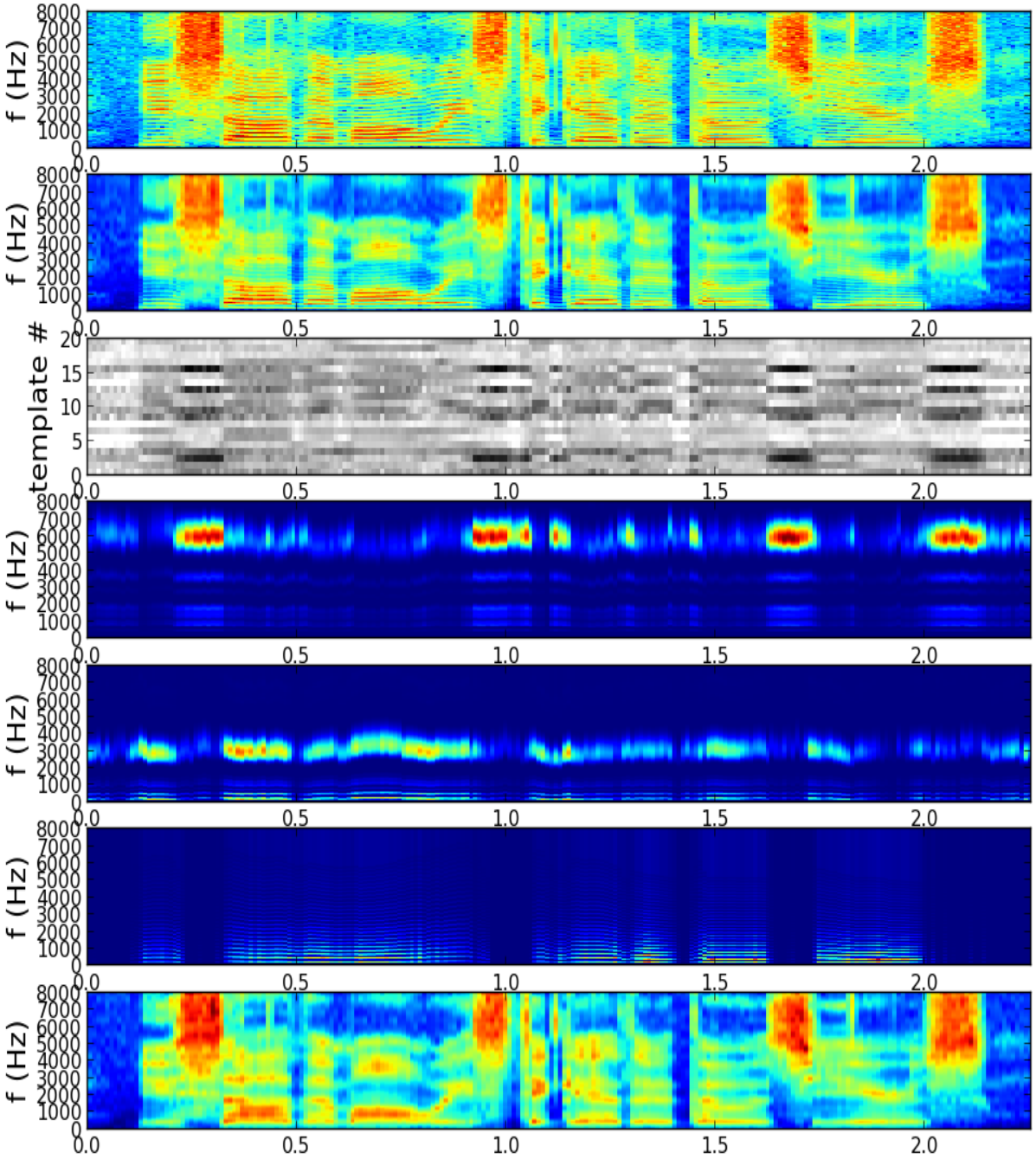


Figure 5.6: Reconstruction of test utterance "You saw them always together those years". using 20 templates learnt on the TIMIT database. *Top row:* Spectrogram of test utterance. *Second row:* Frame by frame reconstruction of test spectrogram using templates. *Third row:* Intensity of the capsules in each of the frames. *Rows 4-6:* Contributions of capsules 3, 7 and 14 to the reconstruction (see figure 5.5a to see the individual patterns - the features are arranged in a column-major order). *Last Row:* Reconstruction of spectrogram from fitting a model of 20 Radial Basis functions to the data. This figure should be viewed in color.

Table 5.1: Impact of number of templates and template sparsity penalty on phone recognition results on TIMIT. DNN-HMM system with two hidden layers without RBM-pretraining were used in each of the following experiments.

# of templates	**λ**	**dev**	**test**
20	0.1	20.3, 20.3, 20.2 (**20.3**)	22.1, 22.2, 22.0 (**22.1**)
	0.2	20.5, 20.3, 20.1 (**20.3**)	22.4, 22.2, 21.6 (**22.4**)
	0.4	20.3, 20.3, 20.3 (**20.3**)	21.9, 22.1, 22.5 (**22.2**)
20	0.1	20.3, 20.3, 20.2 (**20.3**)	22.1, 22.2, 22.0 (**22.1**)
40		20.5, 20.2, 20.3 (**20.3**)	22.1, 22.1, 21.8 (**22.0**)
60		20.2, 20.3, 20.4 (**20.3**)	22.1, 22.0, 21.9 (**22.0**)

# of layers	**frames**		**FBANKS + frames**	
	dev	**test**	**dev**	**test**
2	21.9,22.0,22.0 (**22.0**)	23.6,23.4,23.7 (**23.6**)	19.5,19.7,19.8 (**19.7**)	21.6,21.4,21.8 (**21.6**)
3	21.2,21.4,21.5 (**21.4**)	23.4,22.4,22.9 (**22.9**)	19.0,19.2,19.3 (**19.2**)	20.8,20.9,21.2 (**21.0**)
4	21.2,21.3,21.0 (**21.2**)	22.3,22.1,22.8 (**22.4**)	19.0,19.0,19.0 (**19.0**)	20.7,20.9,20.7 (**20.8**)
5	21.1,21.2,21.3 (**21.2**)	23.1,22.8,22.4 (**22.8**)	19.0,19.0,19.3 (**19.1**)	20.7,20.6,20.7 (**20.7**)
6	21.0,21.4,20.9 (**21.1**)	22.7,22.3,22.3 (**22.4**)	18.8,19.0,18.9 (**18.9**)	20.4,20.6,20.4 (**20.5**)
7	21.2,21.3,20.8 (**21.1**)	22.6,22.1,22.7 (**22.5**)	19.1,19.0,18.8 (**19.0**)	20.5,20.1,20.4 (**20.3**)

Table 5.2: Phone recognition results on TIMIT using templates on frames of spectrograms. RBM pretrained DNN-HMM models of different depths were trained. Each model used 2000 units in every hidden layer. It can be seen that concatenating capsule intensities to Mel log filter banks improves recognition accuracy.

Table 5.2 and figure 5.7 show phone error rate (PER) as a function of depth for the dev and core test set in TIMIT, using filter-banks with template intensities as the feature inputs to the hybrid system. It can be seen that the results improve with depth. The templates help performance at each depth over using FBANKs alone. The dimensionality of the input is 2100 (=1800 + 15 * 20). Each hidden layer used 2000 units. Thus the network with appended features had a slightly larger number of weights (15*20*2000 extra) than our baselines in chapter 2 which is not much considering the total number of weights in these neural networks. Table 5.2 also shows PER achieved when only template instantiation parameters were used for recognition (both instantiation parameters were used). By themselves the instantiation parameters are worse for phone recognition than FBANKs even though the reconstruction of the spectrogram is of very high quality. We surmise that this is possibly because the decoder results in features that are more entangled and less useful for classification.

Table 5.6 shows word error rates achieved when we applied the same method to the 14 hour WSJ corpus *si-84* (see rows using features 'frames' and 'frames + FBANKS'). An autoencoder was trained with 20 templates using the same spectrogram parameters we used for TIMIT. The trained autoencoder was used to compute instantiation parameters on a frame level for WSJ, as was done for TIMIT. These were appended to FBANKS and used for speech recognition. We used a DNN-HMM system with five hidden layers of 2000 sigmoid units. The parameters were pretrained with a DBN. It can be seen that appending these features to FBANKS also produces gains for WSJ over the baseline DNN-HMM system that uses FBANK inputs alone. The gains are larger for the validation dataset than they are for the test data, where the gains are only minimal. This may be because the validation dataset was used to guide the neural network training but may also be the result of the fact that the test database is an easier dataset than the validation dataset. As was seen with TIMIT the results from using template

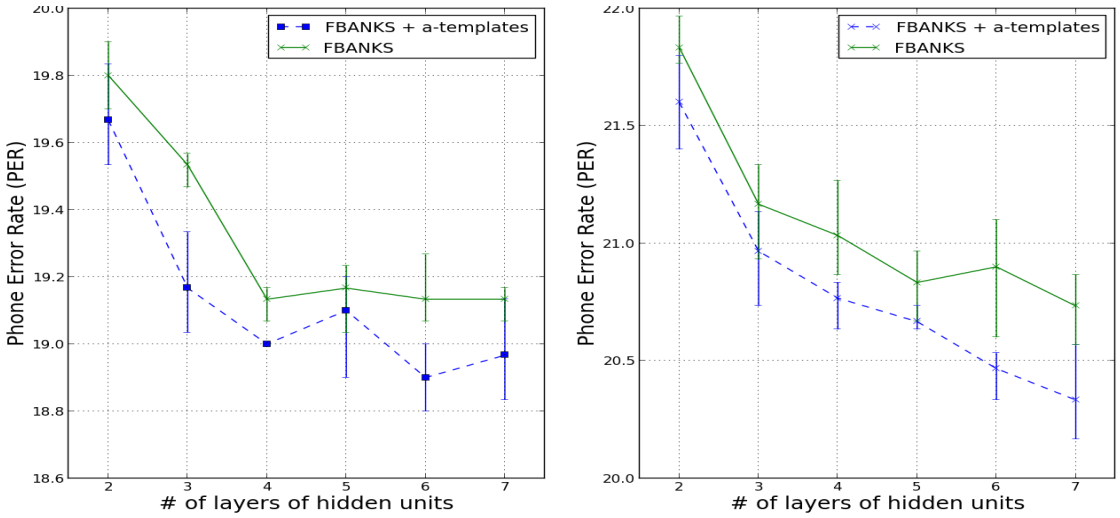


Figure 5.7: Phone Error Rate achieved by appending template intensities to FBANKs on TIMIT (*FBANKS + a-templates*). For comparison we show our baseline results with FBANK inputs to a DNN-HMM system (*FBANKS*). It can be seen that increasing depth improves PER, just as it does for FBANK inputs . The templates lead to improved performance at every depth.

instantiation parameters alone are worse than those from using FBANKs.

5.3 Application to FBANKS Patches

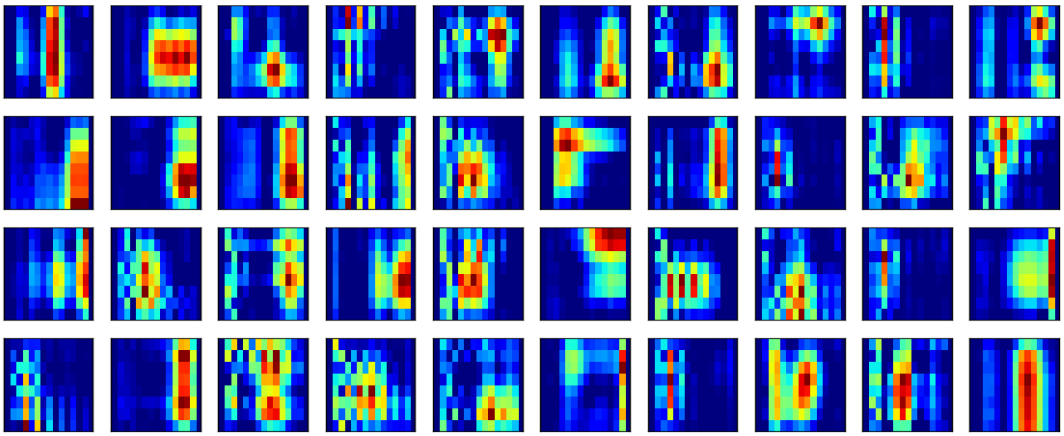
It has recently been shown that convolutional neural networks with convolutions along the frequency domain rather than in the temporal domain can lead to improvements in speech recognition accuracy [Abdel-Hamid et al., 2012]. In this model filters that are local in frequency space and full connected along the 15 temporal frames of input are applied convolutionally along the frequency dimension to FBANK spectra. In the previous section we used an autoencoder that learned to predict instantiation parameters of templates on frames of spectrograms but ignored temporal context. In this section we apply these autoencoders to patches of FBANK spectra rather than individual frames. Using patches over multiple frames allows us to capture temporal aspects in our templates and their instantiation parameters. The instantiation parameters are used for speech recognition in the subsequent section.

5.3.1 Autoencoder Details

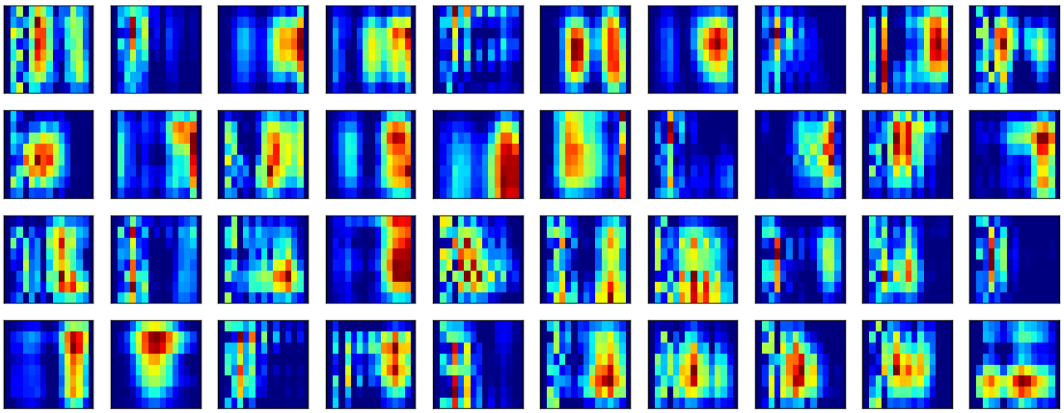
The model used on patches of FBANKs is similar to the model described for images in figure 5.1 and section 5.1. FBANK patches of size 15x40 were used as inputs to the encoder which computes instantiation parameters (x_i, y_i, a_i) $1 \leq i \leq T$ for each of T templates[2]. The decoder shifts the template i, described by template patch $\boldsymbol{\theta_i}$ to location (x_i, y_i) in the reconstruction after scaling the intensity by a_i and adds it to the reconstruction. Learning of the parameters is performed as described in section 5.2.2 using stochastic gradient descent on reconstruction error.

The instantiation parameters x_i, y_i were calculated by rescaling the outputs of sigmoid units to the range of the images. The intensity a_i was computed using a ReLU unit and was used without rescaling.

[2]Note that the height of the patch (=40) is dictated by the dimensions of FBANKs.



(a) *TIMIT.*



(b) *WSJ.*

Figure 5.8: Template features discovered by our method on patches of FBANK spectra of (a) TIMIT (b) WSJ.

5.3.2 Results

Figure 5.8 shows templates that are learnt on FBANK patches of TIMIT and WSJ databases. In this experiment 40 templates of size 13x8 were used to model patches of size 15x40 (=15 frames wide). It can be seen that various templates learn spectro-temporal patterns. For example, the first template in figure 5.5a contributes energies to the full range of 8 vertical/frequency pixels at a narrow range of a few horizontal frames. The second template on the other hand contributes energies over a smaller frequency band on a larger number of frames.

Figure 5.9 shows an application of these templates to the reconstruction of an utterance. It can be seen from the image that the templates are able to reconstruct the utterance quite well. The contributions of three templates to the reconstructions show that templates learn to localize to specific frequency bands rather than over the entire frequency range. The instantiation parameters of the learnt templates were used for speech recognition in the same way as in section 5.2.3. The intensity instantiation parameter of each template was appended to FBANK inputs. Delta and acceleration vectors were used for FBANKs but not for template instantiation parameters. For use in the DNN-HMM system the instantiation

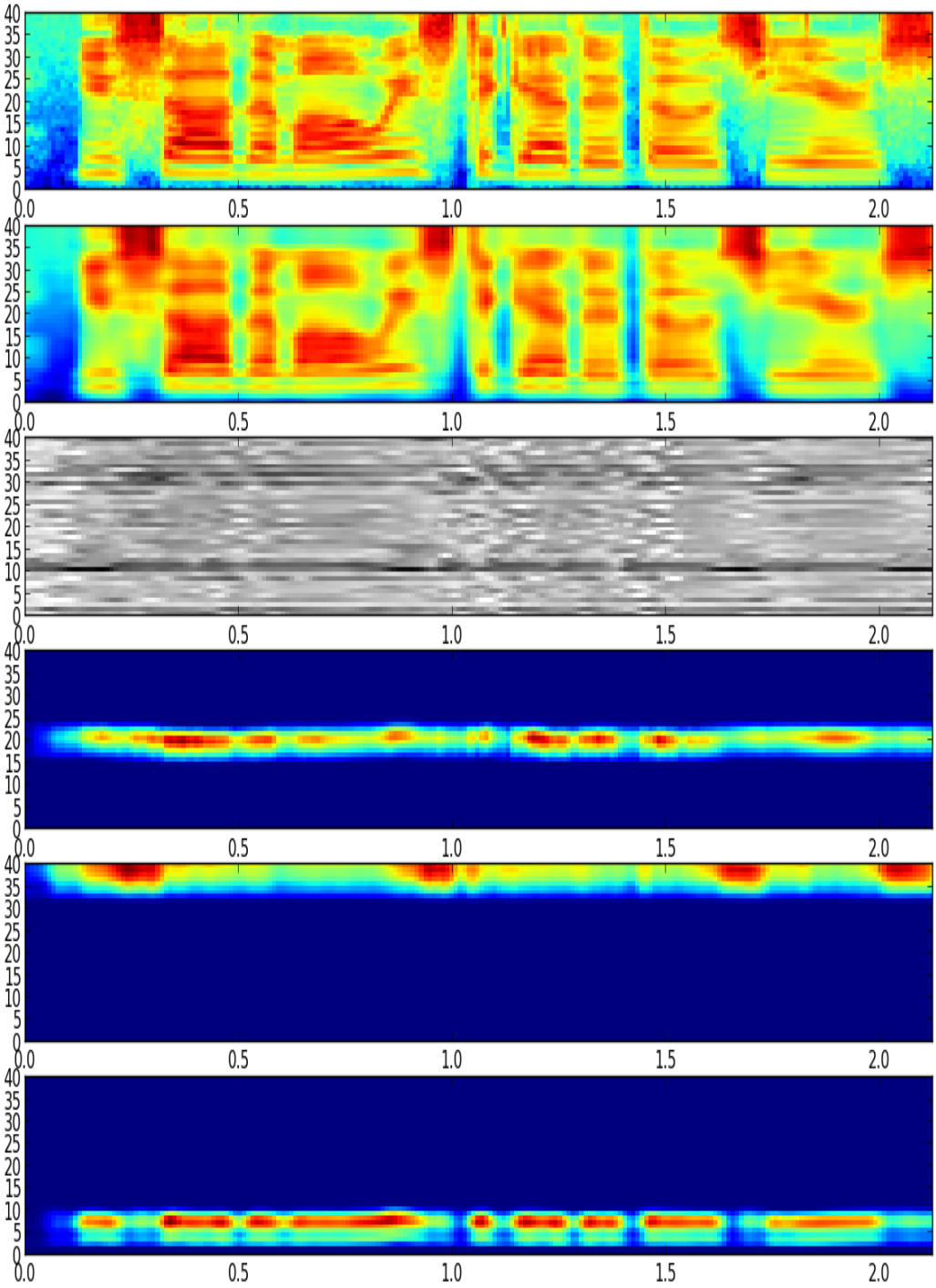


Figure 5.9: Reconstruction of an utterance from TIMIT test database using an autoencoder with 40 templates. The top panel is the original Mel log FBANK spectrum, the second row is its reconstruction. The third row shows the intensities of the 40 templates. The remaining rows are contributions of 3 different capsules to the reconstruction of the utterance.

Table 5.3: Impact of hyper-parameters used for training templates on patches of FBANK spectra on phone recognition results and reconstruction error (on TIMIT). DNN-HMM system with two hidden layers without RBM-pretraining were used in each of the following experiments.

# of frames	# of templates	patch height	patch width	RMSE	PER	
					dev	test
15	40	8	13	0.874	20.4,20.0,20.2 (**20.2**)	21.9,21.9,22.0 (**21.9**)
		10		0.840	19.9,20.1,20.2 (**20.1**)	21.7,22.0,21.8 (**21.8**)
		12		0.837	20.0,20.1,20.4 (**20.2**)	21.7,22.1,22.0 (**21.9**)
		16		0.812	20.2,20.1,20.0 (**20.1**)	21.5,21.9,21.5 (**21.6**)
		20		0.849	20.2,19.9,20.3 (**20.1**)	21.7,21.7,22.4 (**21.9**)
15	40	16	9	0.822	20.0,20.1,19.9 (**20.0**)	21.7,21.7,22.4 (**21.9**)
			11	0.814	20.3,20.0,20.1 (**20.1**)	21.8,21.5,21.8 (**21.7**)
			13	0.812	20.2,20.1,20.0 (**20.1**)	21.5,21.9,21.5 (**21.6**)
30	40	16	20	1.011	20.4,19.6,20.0 (**20.0**)	22.1,21.7,21.3 (**21.7**)
30	40	20	20	1.032	19.6,19.8,19.9 (**19.8**)	21.7,21.5,21.7 (**21.6**)
30	20	20	20	1.245	19.8,20.1,19.7 (**19.9**)	21.9,21.7,22.0 (**21.9**)
	40			1.032	19.6,19.8,19.9 (**19.8**)	21.7,21.5,21.7 (**21.6**)
	60			0.908	19.7,19.7,19.9 (**19.8**)	21.7,21.6,21.6 (**21.6**)

parameter values were rescaled by standard deviation but not mean centered. When the templates were used by themselves for speech recognition, all the instantiation parameters were used.

Table 5.3 shows the impact of different hyper-parameters on the reconstruction error and the phone error rate on TIMIT. The reconstruction error is the root mean squared error between each pixel and its reconstruction using the autoencoder. The DNN-HMM systems used in these tables were not pretrained.

It can be seen that increasing the patch size can improve the reconstruction error however it does not have a very strong impact on PER. Using patches that are 30 frames wide results in worse reconstruction error when the number of templates is kept constant which is not surprising since it has to model a larger patch. However modelling patches that are 30 frames wide does lead to improved phone recognition.

The model learnt over patches of size 30x40 with 20 templates of size 20x20 were used for further experiments in speech recognition with TIMIT. A DBN was pretrained and used to initialize the DNN-HMM systems. Table 5.4 and figure 5.10 show the results obtained for DNN-HMM systems of different depth (the experiments were performed in triplicate). It can be seen that this system outperforms the baseline model at every depth. As before, when only the instantiation parameters are used (see table 5.4), the performance is worse than that of FBANKs, in spite of the good reconstruction. However, these instantiation parameters perform better than the instantiation parameters for individual frames of spectrograms. This is probably because the patch based templates model temporal patterns and also model the characteristics of FBANKS, rather than that of spectrograms — it is well known that the FBANKs are better suited to speech recognition than spectrograms. It is possible that these instantiation parameters would perform better still if they were used within a neural network framework that looks for coincidences between capsules.

A template-based autoencoder with the same configuration as above was also trained on WSJ. A DNN-HMM system was trained using FBANK data augmented with the intensity instantiation parameters similar to the way it was done for TIMIT. Table 5.6 show the word error rates on the validation and test set from three different runs (see rows labelled 'patches' and 'patches + FBANKS'). For the validation set the WER is only slightly better than the baseline with FBANK features. For the test

# of layers	patches		FBANKS + patches	
	dev	test	dev	test
2	20.6,20.7,20.5 (**20.6**)	22.5,22.3,22.5 (**22.4**)	19.4,19.5,19.1 (**19.3**)	21.5,21.3,21.3 (**21.4**)
3	20.0,20.2,20.6 (**20.3**)	21.8,22.2,22.3 (**22.1**)	18.7,18.8,18.9 (**18.8**)	20.9,21.0,20.9 (**20.9**)
4	20.0,19.9,19.7 (**19.9**)	21.7,21.9,21.9 (**21.8**)	18.7,18.6,18.6 (**18.6**)	20.4,20.3,20.8 (**20.5**)
5	19.8,19.9,20.1 (**19.9**)	21.6,21.3,21.8 (**21.6**)	18.7,18.9,18.6 (**18.7**)	20.4,20.2,20.6 (**20.4**)
6	20.1,19.7,19.8 (**19.9**)	21.9,21.6,21.7 (**21.7**)	18.6,18.4,18.4 (**18.5**)	20.4,20.3,20.4 (**20.4**)
7	19.8,19.6,19.7 (**19.7**)	21.7,21.6,21.6 (**21.6**)	18.6,18.5,18.4 (**18.5**)	20.6,20.5,20.4 (**20.5**)

Table 5.4: Phone recognition results on TIMIT using templates on patches of FBANK spectra 30 frames wide. RBM pretrained DNN-HMM models of different depths were trained. Each model used 2000 sigmoidal units in every hidden layer. It can be seen that concatenating capsule intensities to Mel log filter banks improves recognition accuracy.

# of layers	frames+patches		frames+patches+FBANKS	
	dev	test	dev	test
2	20.5,20.3,20.5 (**20.4**)	22.1,22.0,21.8 (**22.0**)	19.2,19.3,19.4 (**19.3**)	21.5,21.3,21.4 (**21.4**)
3	19.6,20.0,19.7 (**19.8**)	21.2,21.5,21.0 (**21.2**)	18.8,19.0,19.3 (**19.0**)	20.7,21.4,20.8 (**21.0**)
4	19.7,19.5,19.6 (**19.6**)	21.6,21.3,21.6 (**21.5**)	18.9,19.0,18.8 (**18.9**)	20.8,20.6,20.7 (**20.7**)
5	19.3,19.8,19.8 (**19.6**)	21.0,21.0,20.6 (**20.9**)	18.9,18.9,18.9 (**18.9**)	20.6,20.3,21.2 (**20.7**)
6	19.5,19.5,19.6 (**19.5**)	20.3,21.1,21.0 (**20.8**)	18.5,18.8,18.8 (**18.7**)	20.7,20.6,20.4 (**20.6**)
7	19.4,19.4,19.3 (**19.4**)	20.5,20.7,21.0 (**20.7**)	18.8,18.8,18.5 (**18.7**)	20.4,20.7,20.4 (**20.5**)

Table 5.5: Phone recognition results on TIMIT after combining templates on frames of spectrograms and patches of Mel log filter banks. RBM pretrained DNN-HMM models of different depths were trained. Each model used 2000 units in every hidden layer. It can be seen that concatenating capsule intensities to Mel log filter banks improves recognition accuracy.

set the WER is slightly worse than that of the baseline. This is in spite of the fact that FER of DNN training improved by a large margin (see section 5.5.1 for a discussion).

Table 5.5 show results when instantiation parameters from templates on frames and templates on patches are combined together, and with FBANK features for phone recognition on TIMIT. When the instantiation parameters from frames and patches are combined together they perform as well as the FBANK features. Further adding FBANKs only improves results slightly — properly controlling overfitting would possibly bring about more improvements. The corresponding experiments were performed for WSJ as well (see rows 'frames + patches' and 'frames + patches + FBANKS' in table 5.6). Again, the templates perform almost as well as FBANKs. Adding FBANKS improves word error rate slightly, even though FER is not improved.

Features	**test_dev93**	**test_eval92**
MFCC	9.7,9.7,9.9 (**9.7**)	6.0,5.8,5.6 (**5.8**)
FBANK	9.3,9.2,9.6 (**9.4**)	5.3,5.4,5.4 (**5.4**)
frames	10.9,10.8,11.2 (**11.0**)	6.2,6.1,6.1 (**6.1**)
patches	10.1,10.2,9.9 (**10.1**)	5.9,5.6,5.9 (**5.8**)
frames + FBANK	9.0,9.0,9.1 (**9.0**)	5.1,5.3,5.4 (**5.3**)
patches + FBANK	9.1,9.1,9.3 (**9.2**)	5.4,5.4,5.6 (**5.5**)
frames + patches	9.6,9.5,9.6 (**9.6**)	5.4,5.4,5.5(**5.4**)
frames + patches + FBANK	9.1,9.2,9.4 (**9.2**)	5.2,5.4,5.3 (**5.3**)

Table 5.6: Word Error Rates on WSJ.

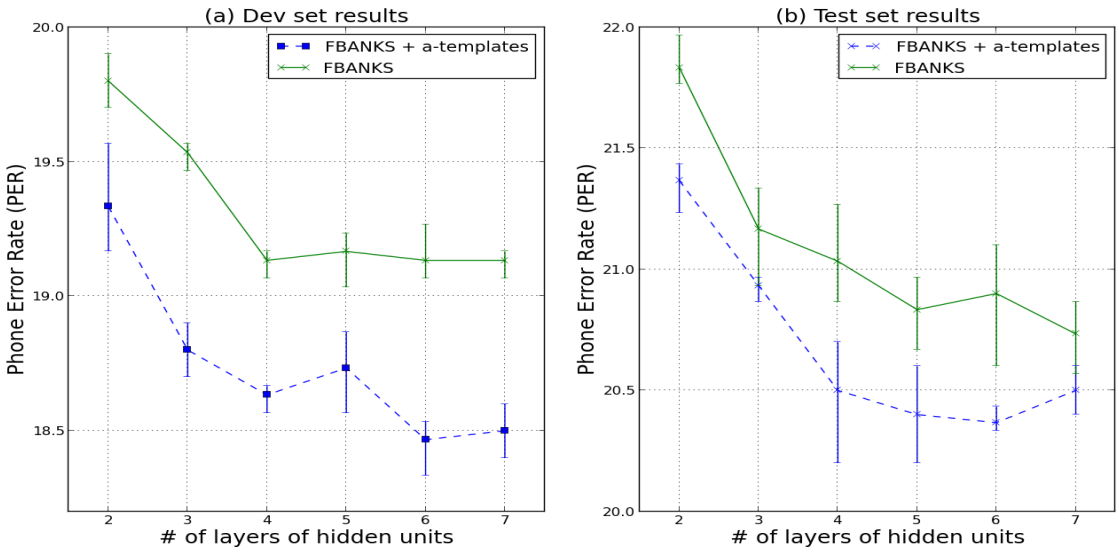


Figure 5.10: Phone Error Rate achieved by appending template intensities of patch based model to FBANKS on TIMIT (*FBANKS + a-templates*). For comparison we show our baseline results with FBANK inputs to a DNN-HMM system (*FBANKS*). It can be seen that increasing depth improves PER, just as it does for FBANK inputs . The templates lead to improved performance at every depth.

# of layers	**2**	**3**	**4**	**5**	**6**	**7**
frames + FBANKs	<0.001	<0.001	<0.001	0.004	0.001	0.014
patches + FBANKs	0.01	∼	0.002	0.019	0.002	∼

Table 5.7: p-values from Matched Pair Sentence Segment Word Error (MPSSWE) test on TIMIT (values greater than 0.05 are labelled as ∼). Each test compares the triplicate results from capsules augmented features to the triplicate results from FBANKs. Results show that augmenting capsules to FBANKs improves results statistically significantly for most depths.

5.4 Significance Testing of Results

We assessed the statistical significance for each of the models using the Matched Pair Sentence Segment Word Error (MPSSWE) test [Gillick and Cox, 1989]. An implementation of this algorithm is available in the program sc_stats that is a part of the NIST Scoring Toolkit (SCTK) (`http://www.itl.nist.gov/iad/mig//tools/`). Each experiment in a triplicate was matched to an arbitrary triplicate of the baseline, and the transcription results were compared using sc_stats. On TIMIT (see table 5.7) we can see that the results were significantly better for most models. On WSJ we found that the results were not statistically significant, even though the frame error rate improvements were very significant ($<$ 0.001).

5.5 Discussion

We have shown how a new type of autoencoder can be used to learn meaningful features called templates on single frames of spectrograms and on patches of FBANK spectra. The instantiation parameters for these templates were concatenated to Mel log filter bank features and used as inputs to hybrid DNN-

dataset	DNN-HMM input	FER	WER
test_dev93	FBANK	41.7,41.8,42.3 (**41.9**)	9.3
	FBANK + frames	39.9,40.0,40.2 (**40.0**)	9.0
	FBANK + patches	39.2,39.9,39.8 (**39.6**)	9.2
test_eval92	FBANK	40.7,40.5,41.2 (**40.8**)	5.3
	FBANK + frames	39.9,39.3,39.2 (**39.5**)	5.2
	FBANK + patches	37.9,37.6,37.9 (**39.8**)	5.5

Table 5.8: Disconnect between frame error rates (FER) of DNN training, and WER of DNN-HMM system for WSJ.

HMM systems. This led to improved speech recognition over DNN-HMM systems using FBANK inputs alone.

However there are some issues that we highlight here that could lead to further improvements in accuracy.

5.5.1 Discord Between Training Objective and Speech Recognition Accuracy

The neural networks using the DNN-HMM systems were trained to predict the state labels from the forced alignments. Each frame in the training set is treated independently even though strong correlations exist from one frame to the next, in both the inputs and the target states. The training can end up ignoring knowledge that is captured in the language model and in the HMM architecture. Even worse, it may miss out on less frequent patterns that are important to distinguish between similar sounds. This would lead to worse speech recognition even with better performance on a frame-based cross-entropy error function.

Table 5.8 shows the FER and the average WER achieved by three different DNN-HMM systems of six layers on three different runs. The first of these systems is our baseline model on si-84 and uses FBANK features as inputs. The other two models are the models trained in this chapter after augmenting FBANK features with frame based and patch based instantiation parameters. It can be seen that the augmented FBANK features show a strong improvement in FER over the baselines. However this does not always translate to much improved WER and may even be associated with worse WER. For example with FBANK inputs, the FER were 41.7, 41.8 and 42.3 for the three runs over the validation set and 40.7, 40.5 and 41.2 on the test set (these numbers are quite consistent — i.e. better FER were almost never observed in my experiments). For the DNN system where FBANK inputs were augmented with patch based features, the FER improved by an average of 1% on the test set (test_eval92) but the WER got worse.

Sequence level objectives for fine tuning DNN-HMM models (see chapter 7 for a summary) are a good way of ameliorating this problem. In chapter 7 we propose another alternative.

5.5.2 More Powerful Decoders

The two decoders we used in this chapter were not very powerful.

The decoder used for frames linearly stretched and compressed the frequency axis. This works for simplistic vocal tract length normalization that assumes the fundamental frequency is inversely proportional to the length of the tube. However, models of vocal tract normalization used for recognition often use a linear warping on the *mel* scale which results in a non-linear warping of the frequency axis

on higher frequencies [Lee and Rose, 1998]. For the speech recognition task, it would have been more suitable to use a decoder that uses instantiation parameters on the mel scale. In addition, it would have been more appropriate to couple together the instantiation parameters of the different templates on a frame - so that a global frequency instantiation was computed, in addition to instantiation parameters that represent deviations from the global mean, for individual templates.

The decoder used for patches placed templates at different rectangular locations in the reconstruction. The templates themselves were quite fixed and can only capture limited variations. Thus interesting patterns such as formant transitions in diphthongs can only be captured through a combination of different templates. Tieleman [2014] explores the use of full affine transformations, however such transitions may be too powerful and almost every template can produce a wide variety of output patterns, eroding the notion of identifiability for such patterns. However a decoder that can transform prototypes over a limited range of distortion may lead to capsules that are identifiable, yet powerful.

5.5.3 Better Use of Instantiation Parameters

In our DNN-HMM system the instantiation parameters were used with the FBANK inputs as-is. However, the instantiation patterns represent spatio temporal characteristics over several FBANK pixels. A better way to use these instantiation parameters may be as gating inputs to the different layers of the neural network. This would allow information over a larger scale of the input to be available to each layer of the network.

Part III

Data Augmentation for Learning Invariances

Chapter 6

Vocal Tract Length Perturbation

This chapter shows a way to build acoustic models that are invariant to variations caused by differences in vocal tract lengths of speakers. Invariance to transformations can either be built into a model through the use of a sensible representation or it can be learnt by providing the model with examples of transformed data. For example, in visual object recognition with neural networks, the use of convolutional layers, makes the model somewhat invariant to the location of the object in the image. Convolutional networks been also been applied to learn acoustic models with a goal of learning an invariance to the precise locations of formants [Abdel-Hamid et al., 2012]. In this chapter, invariance is learnt by neural networks through the use of an augmented dataset created by transforming the original data using different vocal tract lengths. This strategy benefits both fully connected and convolutional neural network acoustic models [Jaitly and Hinton, 2013b]. Kanda et al. [2013] explore a very similar idea and show that it leads to significant improvements in word recognition on a database of Japanese lecture recordings.

6.1 Introduction

Data augmentation is a key ingredient of the state of the art systems for object recognition [LeCun et al., 1998, Krizhevsky et al., 2012, Simard et al., 2003, Ciresan et al., 2011]. In these systems the data vectors are transformed in such a way as to preserve labels. For vision systems such transformations are easy to conceive of - translating, rescaling and distorting the image locally, etc, are transformations that do not typically affect the class label of the object in the image, but are yet able to create new related data examples. Data augmentation is very useful for small datasets where the number of examples is low, such as MNIST [LeCun and Cortes, 1998]. However, it has also been helpful in medium sized datasets such as ImageNet [Deng et al., 2009]. With the widespread adoption of neural networks in speech recognition systems it is natural to ask if the same strategy could be applied here. In this chapter we show that it is indeed possible to augment speech databases, and to use the augmented database to achieve improved accuracy.

Voice Conversion [Ye and Young, 2003, Abe et al., 1988, Kain and Macon, 1998, Stylianou et al., 1998, Toda et al., 2007, Helander et al., 2012] is a natural candidate for generating transformations of utterances to augment a database used for training acoustic models. Each utterance could be transformed to generate new training utterances with characteristics of new speakers. A learning algorithm could learn an invariance to speakers by using the augmented database for training. However, typical voice

conversion algorithms require several utterances per speaker and the number of target speakers is limited to a few chosen speakers for whom good models can be developed. Applying voice conversion to generate an augmented database is, therefore, not a very attractive option at this time.

An alternative method for generating variations in input data is to apply speaker normalization methods in the reverse manner - i.e. instead of using normalization methods to remove speaker to speaker variations, we can use them to *add* variations of input data to the training set. This can be done by *perturbing* input data to random targets, instead of normalizing them to a canonical mean. In this chapter, we show that Vocal Tract Normalization (VTLN) techniques can be used successfully for this.

VTLN is used in speech recognition to remove speaker to speaker variability that results primarily from differences in vocal tract length [Lee and Rose, 1998]. To achieve vocal tract normalization, the frequency axis of the spectrogram of each speaker is linearly warped using a warp factor α, that accounts for the relative length of their vocal tract compared to the canonical mean. This factor is estimated for each speaker in the training database during training with GMM-HMMs. For speakers in the testing database, different strategies may be applied to fit the warp factor during decoding [Lee and Rose, 1998].

In this chapter we augment the training database by generating a random warp factor for each utterance during training. We call this Vocal Tract Length Perturbation (VTLP). We show that this method leads to consistent gains across DNN-HMMs of different depths, and on different datasets. For TIMIT this method produced average gains in PER of 0.4% to 0.7% on the dev set, when using DNNs of different depth. On the test set it produced average gains from 0.1% to 0.4% over the same experiments. VTLP also produced a gain of 1.0% on the test set when a convolutional neural network (CNN) was trained. On WSJ-si84 the strategy resulted in an average gain of 0.6% WER on the development set (test-dev93) and a corresponding average gain of 0.4% WER on the test set (test-eval92) [1]

When different random perturbations were generated on the basis of gender, further improvements in results were obtained. On TIMIT, the strategy produced gains of 0.4% to 0.9% on the dev set, and 0.4% to 0.8% on the test set. On WSJ-si84 an average gain of 0.6% and 0.5% were achieved in the WER of development and test sets respectively.

Data augmentation through transformation, therefore, represents an important direction to be exploited, especially for projects, such as BABEL (`http://www.iarpa.gov/Programs/ia/Babel/babel.html`), where the amount of transcribed training data is limited.

6.2 Methods

We first summarize how Mel log filter bank features are computed. Then we describe how we generate transformed utterances using VTLP and Mel filter banks. Lastly, we discuss how deep neural networks were trained and used during decoding.

6.2.1 Mel Filter Banks

It is well known that the human auditory apparatus does not perceive pitch on the same scale as the frequencies of sound in Hz [Licklider, 1951]. In a landmark study that is as remarkable a read today as it must have been over 75 years ago Stevens et al. [1937] discovered a scale for perceived pitch using a

[1] Note that all numbers are absolute improvements over baselines, not relative improvements.

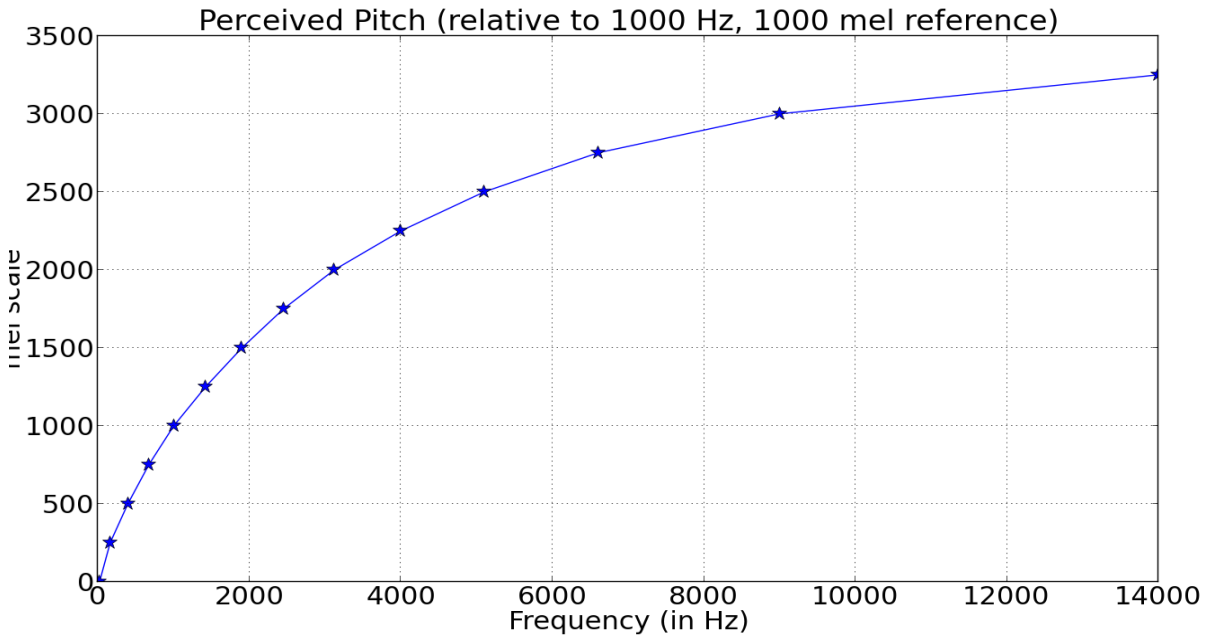


Figure 6.1: Plot of perceived pitch vs frequency from perceptual data acquired in Stevens et al. [1937] as compiled by Beranek et al. [1988].

very clever device. Stevens et al. [1937] presented 10 observers with the task of dividing three different frequency ranges (40-1000Hz, 200-6500Hz and 3000-12000Hz) into five equally spaced intervals using a specially built electronic device. The device had three different knobs, each controlling the frequency of a tone. By turning these knobs the observers could adjust the frequency to match the three perceived internal quantiles of the frequency range. It was observed that there was remarkable consistency within individuals with regards to the frequency that is perceived to be half the pitch of another frequency. Figure 6.1 shows a plot of these experiments as compiled by Beranek et al. [1988]. It is clear from these curves that the human auditory apparatus has higher resolution in lower frequencies and lower resolution in higher frequencies. Different formulae have been proposed to fit this scale (e.g. [Koenig, 1949, Fant, 1967]) but the following function by Makhoul and Cosell [1976] is the one that is widely used in speech recognition systems today:

$$m(f) = 1127.01 * \log\left(1 + \frac{f}{700}\right) \tag{6.1}$$

Speech recognition systems warp spectrograms using this Mel scale to match the human auditory apparatus. In addition to warping to the Mel scale, the systems also take into account two additional properties of the human auditory apparatus - the bandwidth of an auditory fiber and its masking pattern. Each auditory fiber responds to a range of frequencies around a center frequency that it is most sensitive to Fletcher [1940], Patterson [1976]. The shape of filter response pattern and the bandwidth of the response has been an area of extensive research (see Moore and Glasberg [1983] for a review). Speech recognition systems typically assume that each auditory filter has a triangular response shape around the center frequency and the width of filter is determined by the center frequencies of its neighboring filters [Greenberg et al., 1986]. This set of filters is known as the Mel filter banks.

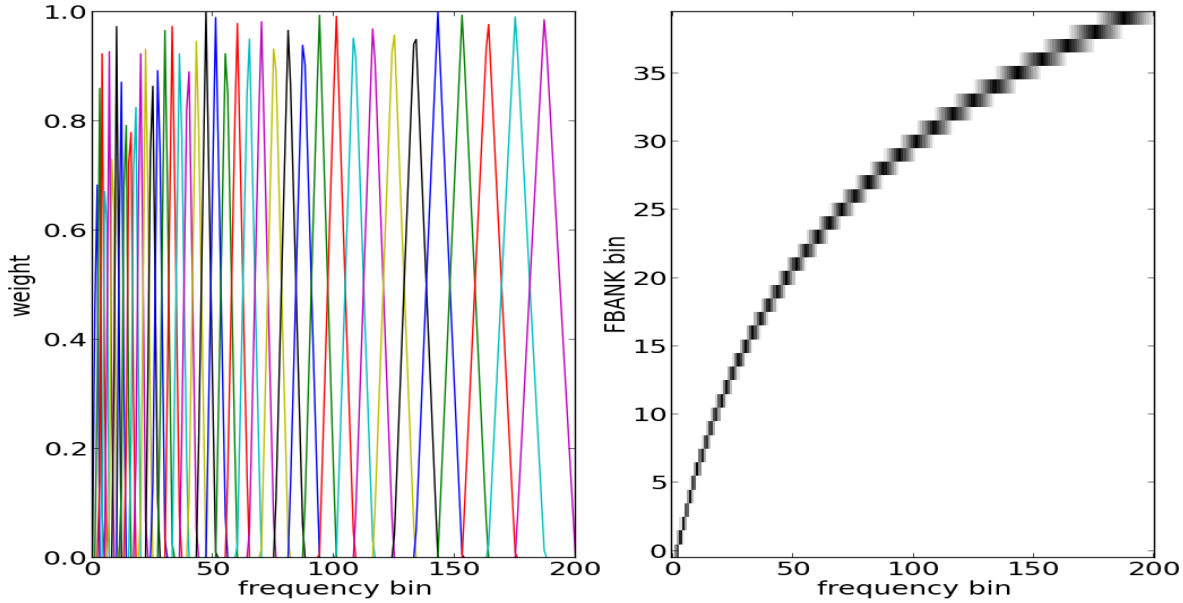


Figure 6.2: Two views of Mel filter banks. *Left:* Line plot of filter banks. Each filter bank assigns triangular weights to the different input frequency bins. The center frequency of of one filter bank is the left edge of the previous filter bank and the right edge of the next. *Right:* Matrix showing the weights connecting input frequencies (x-axis) to the output filter bank bins (y-axis). It is clear that Mel filter banks represent a very structured, sparse linear operation.

In this scheme, a set of N filters are centered at, equi-distant points on the Mel scale. Thus for N filter banks, between a frequency range of F_{min} and F_{max}[2], the center frequency of the i^{th} filter bank is:

$$f(i) = m^{-1}\left(m\left(F_{min}\right) + \frac{m\left(F_{max}\right) - m\left(F_{min}\right)}{N-1}\left(i-1\right)\right) \tag{6.2}$$

where $m^{-1}(s) = 700\left(\exp\left(\frac{s}{1127.01}\right) - 1\right)$ is the inverse of the Mel scale function. Each filter bank starts from the center of the previous filter bank and ends at the center of the next filter bank. We used a triangular window function for each filter bank, with the max value of 1 at the center and 0 at the boundaries.

The filter banks are applied to either the power or the absolute magnitude spectra of speech signals, and log transformed [3]. Thus Mel log filter bank features are computed using a linear transform followed by a non-linear operation, very much like layers in a neural network. Figure 6.2 shows two different views of Mel filter banks. On the left is the typical plot of FBANKs where each filter is shown as a line plot of a different color. The triangular profile of the filter shows clearly in this view. On the right we show Mel filter banks as a matrix, to underscore the view that it is a linear operator.

[2]We used $F_{min} = 30$Hz and $F_{max} = 8000$ Hz although, $F_{min} = 300$ may be more typical.
[3]We used the power spectrum in this chapter.

6.2.2 VTLN

Vocal Tract Length normalization attempts for correct for the difference in lengths of vocal tracts of individuals. Because the human vocal tract is fundamentally a tube with an open end, the frequency characteristics of sounds that are produced are directly influenced by the length of the vocal tract. This is, of course, more accurate for voiced sounds where the sound wave travels through the entire vocal tract than it is for fricatives and consonantal sounds where turbulent flow close to the lips shapes the sound characteristics. For voiced sounds, however, the resonant frequencies are directly influenced by the length of the vocal tract. Increasing vocal tract length has an inversely proportional effect on the resonant frequencies - causing a linear contraction or expansion of the frequency axis.

Lee and Rose [1998] suggested a way to rescale the frequency axis that diminishes the impact of boundary frequencies. We use the rescaling of the frequency axis as implemented in Povey et al. [2011b]. A rescaling factor warp factor α maps a frequency f to a new frequency f' as follows:

$$f' = \begin{cases} F_{min} + \frac{\alpha f_{lo} - F_{min}}{f_{lo} - F_{min}}(f - F_{min}) & f < f_{lo} = max(F_{lo}, \frac{F_{lo}}{\alpha}) \\ \alpha f & f_{lo} \leq f \leq f_{hi} \\ F_{max} + \frac{F_{max} - \alpha f_{hi}}{F_{max} - f_{hi}}(f - F_{max}) & f > f_{hi} = min(F_{hi}, \frac{F_{hi}}{\alpha}) \end{cases} \tag{6.3}$$

where F_{min} and F_{max} are the minimum and maximum frequencies in the spectrogram, F_{lo} and F_{hi} are boundary frequencies that are chosen such that the significant formants lie within their range. We used $F_{lo} = 300$ and $F_{hi} = 5000$. Here α can be interpreted as the ratio of the vocal tract of a canonical reference to the vocal tract of the length of the individual that produced the utterance.

This rescaling of the frequency axis has a non-linear impact on the Mel filter bank spectrum that is computed. VTLN is a way to build invariance into the features by learning a warp factor for each speaker or even each utterance.

The rescaling of the frequency axis may seem computationally intensive but can actually be achieved simply by warping the centers of the Mel filters during the computation of Mel filter banks. Thus $f(i)$ of the $1 \leq i \leq N$ filter banks are remapped to $f'(i)$, using equation 6.3 and triangular filter banks are generated at these warped frequencies. This avoids the extra computation needed to rescale high dimensional spectrograms.

Figure 6.3 (a) shows the mappings produced by three different warp factors, $\alpha = 0.8, 1, 1.2$. Figure 6.3 (b) shows the corresponding filter bank projection matrices. Clearly VTLN can be thought of as a neural network layer whose parameters lie on a simple manifold defined by the warp factor α. VTLP is an attempt to learn a model that is invariant to the choice of the warp factor.

6.2.3 VTLP

To encourage models to be invariant to the choice of the optimal warp factor for each utterance, at training time we compute Mel log filter bank spectra using random warp factors. In VTLN approaches the warp factor is typically assumed to lie between 0.8 and 1.2. Since our goal was not to normalize, but to introduce variation, we chose a smaller range of 0.95 to 1.05. A larger range may create unrealistic distortions by distorting utterances with actual high warps or low warps beyond the boundaries of 0.8 and 1.2.

Two different perturbation strategies were explored:

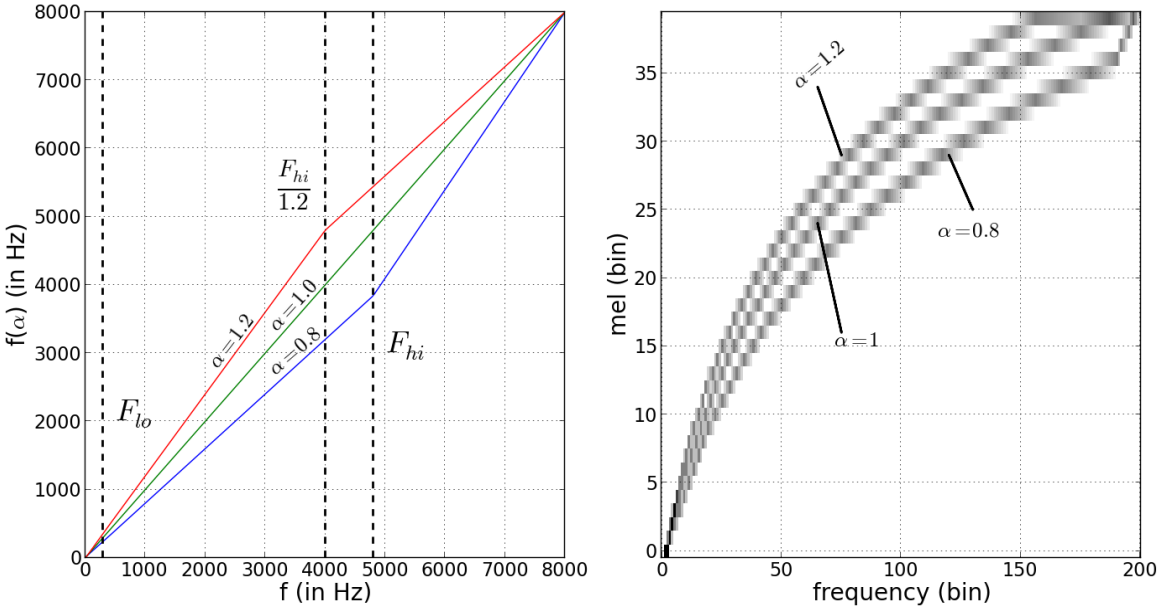


Figure 6.3: (a) Plot of different warps of the frequency axis. (b) Overlaid plot of Mel filter bank projection matrices at three different warp factors.

- *Uniform warps:* Warp factors were uniformly generated for each utterance within the range 0.95-1.05 for each utterance at the start of an epoch.

- *Gender specific warps:* Warp factors were generated from a truncated normal distribution with standard deviation of 0.1 with a mean that depends on the gender of the speaker; for male speakers a mean of 1.05 was used whereas for female speakers a mean of 0.95 was used.

6.2.4 Neural Network Training

At the start of each epoch of neural network training we warped each utterance with a random warp factor α generated using the strategies described above. 40 dimensional log Mel-filter bank coefficients were computed and delta and acceleration parameters were appended to each frame. Mean and variance were computed over the entire database using random perturbations for each utterance. These were used to standardize the 120 dimensional inputs from each frame. A context of +/-7 frames was used as the input to a neural network to predict the phoneme state label of the center frame from forced alignment.

6.2.5 Decoding

Once a neural network has been trained with randomly transformed inputs, its predictions can be used for decoding in several ways.

VTLN techniques typically fit a warp factor to each speaker that gives rise to the best log likelihood of the utterances for that speaker from the GMM-HMM model [Lee and Rose, 1998]. This warp factor would then be used for decoding the utterances from that speaker. But this requires extra enrollment

utterances from test speakers to learn a warp factor. Since the goal of this thesis is to investigate speaker independent methods, this is not optimal.

Alternatively, we may apply this strategy on an utterance level - a warp factor would be fit to each utterance separately without considering speaker identity. This would be done during a preliminary decoding phase. Subsequently, the learnt warp factor could be used in the neural network. However, this requires the use of a trained GMM model and multiple rounds of decoding.

Instead we approached the problem from a model averaging perspective. At test time, V variant vectors were generated for each utterance by using equally spaced VTLP warp factors between 0.95 and 1.05. The predicted posterior probabilities of the HMM states were obtained by performing a forward pass through the neural network, for each of these variants. These multiple predictions were combined by the following methods:

- *Mean:* Predicted probabilities were averaged for each frame separately.
- *Geometric:* Predicted probabilities were geometrically averaged. This is equivalent to averaging the logits of the softmax layer.
- *Maximum Confidence:* The entropy of the predictions over all the frames of an utterance was computed for each of the candidate warp factors. The warp factor that led to the most confident predictions (i.e. with least entropy) was chosen as the single warp factor to be used for that utterance.

The merged probabilities were then used in a DNN-HMM hybrid system to perform decoding. It should be noted that these decoding strategies do not use gender information even for models that were trained with gender-specific VTLP.

6.3 Experiments

We used the baseline recipes from chapter 2 to generate alignment labels for utterances of TIMIT and WSJ-si84. The spectrograms for these utterances were computed using the same window and stride intervals that were used in the baseline - 25 ms interval windows with a stride of 10 ms. Spectrograms were thus 201 dimensional, since the FFTs were computed over 400 samples of raw signal. 40 dimensional Mel scaled filter banks were generated. Deltas and accelerations were appended (actually we striped the coefficients so that filter bank values and their deltas and accelerations were in adjacent indices - this makes it easier to code up the CNNs, which have a built-in assumption of pixel locality). These were normalized as described above. For WSJ each speaker was mean centered and normalized separately, as was done for the baselines.

6.3.1 Fully Connected DNNs

We trained fully connected DNNs of depths from two to seven layers The layers were pretrained using parameters of a Deep Belief Network (DBN) [Hinton et al., 2006, Hinton and Salakhutdinov, 2006]. The DBN was trained using layer by layer training of Restricted Boltzmann Machines (RBMs) [Smolensky, 1986] as described in Hinton et al. [2006]. The bottom layer was used to model the data whereas the subsequent layers modelled the distribution of the hidden activities of the layer below, conditioned on the data. A Gaussian-Binary RBM [Welling et al., 2004] was used at the bottom layer to model the

data and Binary-Binary RBMs were used in the layers above. The data used to train the bottom layer was generated using the same scheme that is used for the neural networks - at the start of each epoch, a random warp factor is applied to each utterance in the database. These are normalized and used as inputs to the DBNs. Two 7 layer DBN were thus trained - one for each of the perturbation strategies - using 2000 binary units in each layer. The learnt parameters were used to pretrain each of the layers of all the DNNs from depth two to seven.

The DNNs used sigmoid units in each layer. Learning of neural network parameters was performed using back propagation with stochastic gradient descent and momentum. A learning rate of 0.1 was used in the first epoch; At the end of each epoch if the prediction accuracy over the state labels for dev set frames did not increase the learning rate was annealed by a factor of 0.5 and the parameters were reset to their values at the start of the epoch [4]. A momentum of 0.0 was used for the first epoch, and 0.9 for the rest. Note that these hyper-parameters are the same ones we used for our baselines.

The gender specific models were trained in the same way.

6.3.2 Convolutional Neural Network (CNN)

The CNN we used had a similar architecture to Abdel-Hamid et al. [2012]. 91 local filters that looked at 8 consecutive frequency bins (and associated deltas and accelerations) over all the 15 temporal frames were applied at 6 consecutive frequency bins. These values were max-pooled together. 20 such different sets of filters were applied starting at each even numbered bin. The max-pooled activities were passed through a Rectified Linear Unit (ReLU) Nair and Hinton [2010]. Thus there were 1820 (=91*20) units in this layer. There were two additional layers of fully connected hidden units of 1000 ReLUs, each before the output layer. During back propagation through the convolutional layers, the gradient is only computed for those units that were chosen as the max units in the pooling.

The hyper-parameters for the CNNs were chosen by cross-validation with the validation set in preliminary experiments. A learning rate of 0.03 was used in the first epoch and annealed by 0.9 every time the frame error rate increased over the previous epoch. A momentum of 0.8 was used for the first epoch and 0.9 thereafter.

6.4 Results

Figure 6.4 shows a plot of the PER of DNNs of depth two to seven trained on TIMIT. Three DNNs were trained for each depth. Training was done using warp factors uniformly drawn between 0.95 and 1.05. The utterances from the test and dev sets were decoded using a warp factor of 1.0. It can be seen that VTLP leads to consistent gains for each of the layers.

Figure 6.5 shows a plot of the PER from the same DNNs using the different decoding strategies described in section 6.2.5 and decoding with a warp factor of 1. Averaging during decoding was done using three warp factors - 0.95, 1.0 and 1.05. On the dev set the *minimum entropy* strategy leads to best results, whereas *geometric* and *mean* strategies lead to smaller gains over the decoding with warp factor of 1. On the test set the *minimum entropy* decoding also leads to better results on average than the other decoding strategies but the gains are not consistent over the DNN's of different depths. Table 6.1 show the results of the triplicate experiments using the *minimum entropy* decoding strategy.

[4] Note that this is different from Abdel-Hamid et al. [2012], Mohamed et al. [2012] where annealing is done based on the PER from decoding the dev set

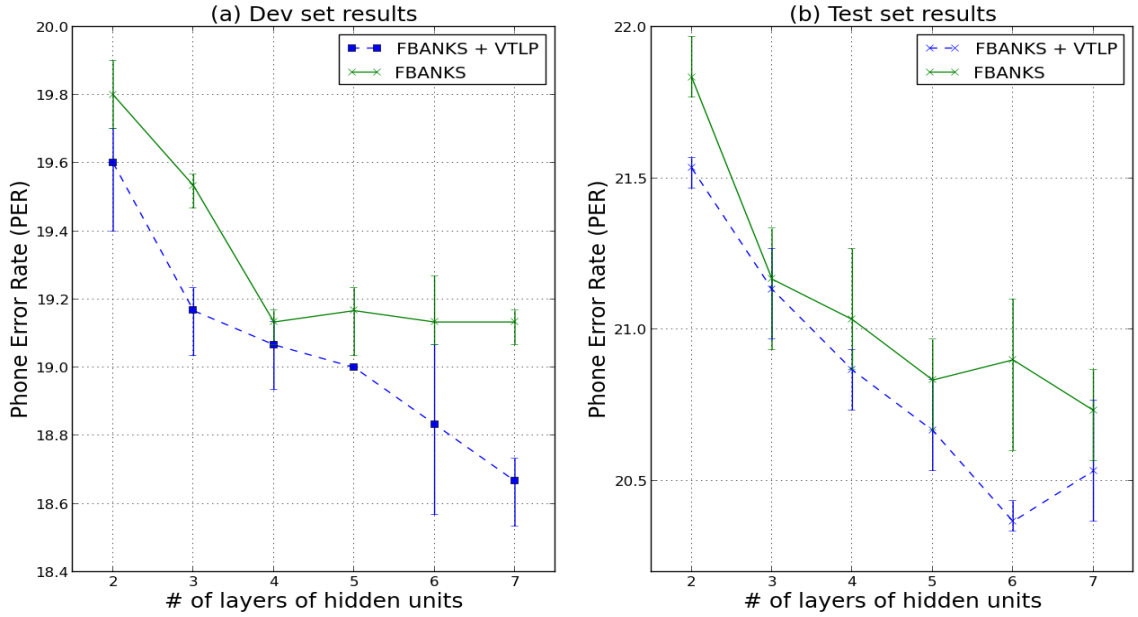


Figure 6.4: Plot of PER vs depth achieved by DNN-HMM's on TIMIT using VTLP. **Left:** Line plot of PER for *dev* set. **Right:** Line plot of PER for *test* set. The model trained with VTLP inputs was decoded using a warp factor of 1 for this figure.

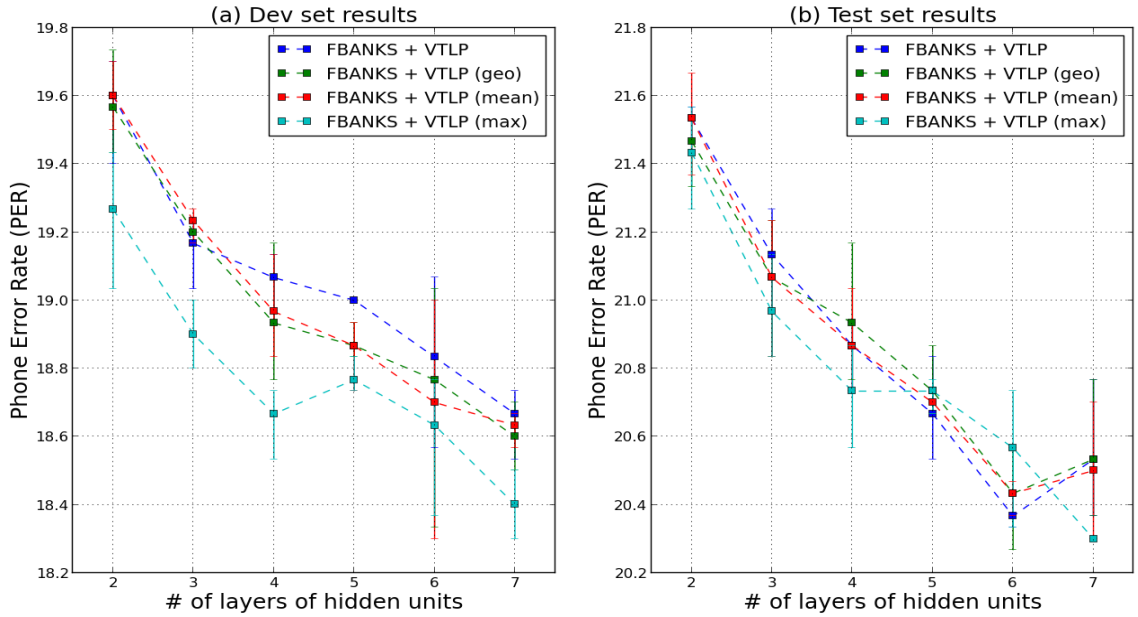


Figure 6.5: Plot of PER vs depth achieved by DNN-HMM's on TIMIT using VTLP with different averaging strategies at test time. **Left:** Line plot of PER for *dev* set. **Right:** Line plot of PER for *test* set. The model *FBANKS + VTLP* was decoded using a warp factor of 1. *FBANKS + VTLP (geo)* used a geometric average of predictions with different warp factors. *FBANKS + VTLP (avg)* used an arithmetic average of predictions with different warp factors. *FBANKS + VTLP (max)*) used predictions from a single warp factor that had minimum entropy

# of layers	dev	test
2	19.1,19.5,19.2(**19.3**)	21.4,21.6,21.3(**21.4**)
3	19.0,18.9,18.8(**18.9**)	20.8,21.1,21.0(**21.0**)
4	18.6,18.8,18.6(**18.7**)	20.9,20.6,20.7(**20.7**)
5	18.7,18.8,18.8(**18.8**)	20.7,20.8,20.7(**20.7**)
6	18.5,18.5,18.9(**18.6**)	20.5,20.8,20.4(**20.6**)
7	18.3,18.4,18.5(**18.4**)	20.3,20.3,20.3(**20.3**)

Table 6.1: Phone recognition results on TIMIT using uniform Vocal Tract Length Perturbation. RBM pretrained DNN-HMM models of different depths were trained. Each model used 2000 sigmoidal units in every hidden layer.

# of layers	dev	test
2	19.5,19.5,19.2(**19.4**)	21.6,21.4,21.3(**21.4**)
3	18.8,18.7,19.0(**18.8**)	21.0,20.4,21.0(**20.8**)
4	18.4,18.7,18.6(**18.6**)	20.5,20.2,20.5(**20.4**)
5	18.3,18.3,18.3(**18.3**)	20.1,19.8,20.1(**20.0**)
6	18.5,18.4,18.5(**18.5**)	20.3,20.2,20.1(**20.2**)
7	18.3,18.3,18.3(**18.3**)	19.9,20.0,20.3(**20.1**)

Table 6.2: Phone recognition results on TIMIT using gender specific Vocal Tract Length Perturbation at training time. RBM pretrained DNN-HMM models of different depths were trained. Each model used 2000 sigmoidal units in every hidden layer. At test time a warp factor is chosen for each utterance that leads to the lowest entropy in the output distribution.

Figure 6.6 shows a plot of the PER using different ways of generating warp factors. It is clear that generating gender specific random warp factors leads to significantly improved PER over the uniformly generated warp factors for both the dev and test sets. On the dev set average gains between 0.4% to 0.9% were observed over the FBANK baselines. On the test the average gains ranged from 0.4% to 0.8%. For these experiments the decoding for the VTLP models was again done with *minimum entropy* strategy over three warp factors - 0.95, 1.0 and 1.05. Table 6.2 shows the values of the PER for the DNN models trained with gender specific warp factors. Note once again that at test time gender information is not used - the appropriate warp factor is selected on the basis of the certainty of the model.

Table 6.3 shows the WER results from 6 layer models that were trained with WSJ-si84. Average WER of the uniform VTLP model decreased by 0.6% absolute on test-dev93 and by 0.4% absolute on test-eval92 compared to the baseline 6 layer model in chapter 2. Correspondingly, the gender specific VTLP improved WER by 0.6% on test-dev93 and by 0.5% on test-eval92. Clearly for WSJ the gender specific perturbation does not lead to as much gain over the uniform perturbation as it does for TIMIT.

For the convolution neural network, the baseline PER was 21.7%. With VTLP an improvement of 1.0% was achieved.

Training Inputs	dev	test
MFCC	9.7,9.7,9.9 (**9.7**)	6.0,5.8,5.6 (**5.8**)
FBANK	9.3,9.2,9.6 (**9.4**)	5.3,5.4,5.4 (**5.4**)
VTLP (max)	9.1,8.7,8.7 (**8.8**)	5.1,4.9,5.0(**5.0**)
VTLP (m/f)(max)	8.9,8.8,8.8 (**8.8**)	4.9,4.8,5.1 (**4.9**)

Table 6.3: WER of DNN-HMMs system on WSJ 14 hour subset (*si-84*) using VTLP.

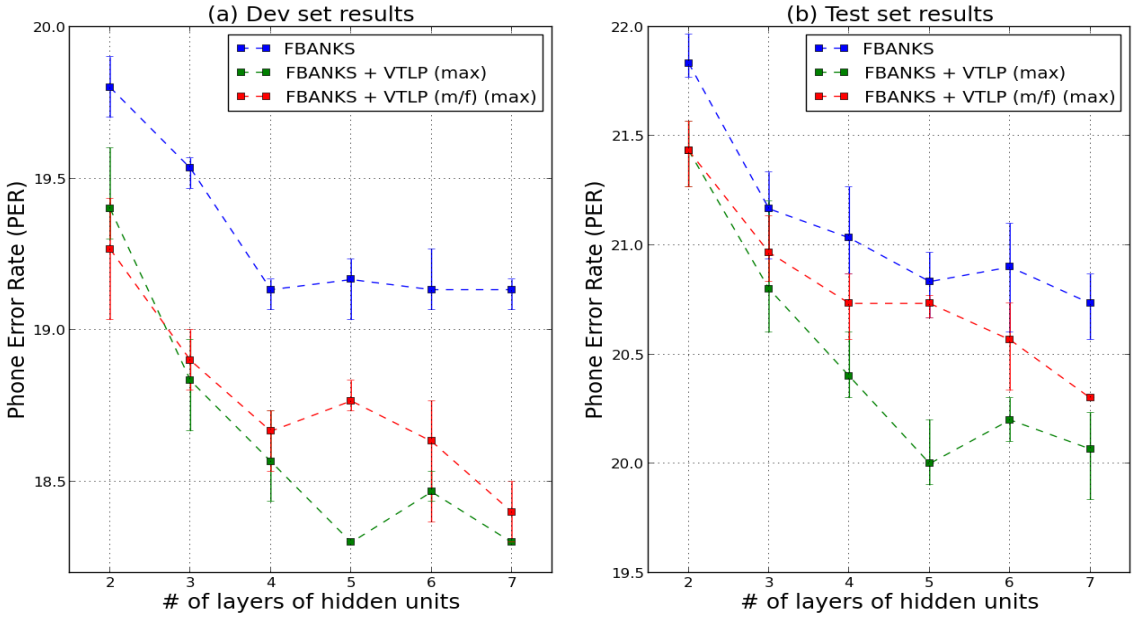


Figure 6.6: Plot of PER vs depth achieved by DNN-HMM's on TIMIT using gender specific VTLP. **Left:** Line plot of PER for *dev* set. **Right:** Line plot of PER for *test* set. The model *FBANKS + VTLP* was decoded using a warp factor of 1. *FBANKS* is the baseline model. *FBANKS + VTLP (max)* is the VTLP model without gender specific distortion using minimum entropy decoding. *FBANKS + VTLP (m/f) (max)* is the gender specific VTLP model using minimum entropy decoding.

6.4.1 Significance Testing of Results

We ran Matched Pair Sentence Segment Word Error (MPSSWE) test to assess for statistical significance of difference. All the results on TIMIT and WSJ were significant at p-value less than 0.01.

6.5 Discussion

It must be noted here that the strategy of generating perturbations to augment the training data is very commonly used in vision. However, it seems to have been ignored in speech. It is possible that this may be due to the difficulty in finding a reasonable transformation function. In fact we have tried several transformations that seem to have been ineffective in providing improvements - warping log spectrograms and using them as input to neural networks, warping Mel log filter bank spectra along the frequency domain and the time domain, applying small changes to LPC coefficients and reconstructing wav files, etc. It is our opinion that perturbation is most effective when applied to the linear spectrograms before the application of filter banks. In addition, perturbation of the input by itself is not as effective without averaging the predictions over multiple perturbations of the data. The two strategies need to be combined for best results.

The improvement for gender specific VTLP over uniform VTLP was much larger for TIMIT than it was for WSJ. This is probably because it is easier for it to learn invariance from the larger WSJ training set without the injection of extra information.

It is interesting that VTLP helps even with WSJ-si84, which is a larger corpus of 14 hours. However on a single experiment with the 81 hour set (WSJ-si284) we found little improvement over a baseline system with the same set (WER=7.6% with VTLP vs. 7.7% without it). This not very surprising - VTLP acts as a regularizer and regularizers are most useful for small data collections.

6.6 Conclusions

In this chapter we have shown that generating random, linearly warped variants of spectral data can be used to enhance recognition accuracy significantly. We have shown how improved decoding results can be achieved by combining multiple predictions over the same data. Improvements were seen even on WSJ-si84 which is a larger database. However, we have only scratched the surface of the variations that can be applied. Naturally, temporal distortions can be applied to spectrograms. Other distortions, such as non linear distortions can also be applied along the frequency dimension (probably most effectively, if applied before multiplication with filter banks).

Part IV

Partial Structure Learning

Chapter 7

Using Long Range Predictions

The previous chapters of the thesis focussed on learning features from acoustic data. These features were used to predict the phone sub-state corresponding to the acoustic data. However, a real speech recognition system aims to minimize the errors in the word level transcripts it produces and training data frames independently from forced alignments ignores this sequential objective. In this chapter we explore an alternative way of training our neural networks that incorporates sequential structure.

Sequence discriminative training is a class of algorithms used to train GMM-HMMs to output correct sequences given acoustics, rather than improving the probability of acoustic data given the utterance - as Maximum Likelihood Estimation with the Expectation Maximization algorithm is apt to do [Valtchev, 1995, Povey, 2004, McDermott et al., 2007]. Maximum Mutual Information (MMI) is a popular sequence discriminative method that was first proposed by Bahl et al. [1986]. The algorithm works by improving the posterior probability of the correct transcript while decreasing the posterior probability of incorrect transcripts. The exact posterior probability is computationally expensive to compute for large vocabulary continuous speech recognition (LVCSR) because the HMMs for LVCSR tasks can have millions of states and have sparse connectivity that makes it computationally infeasible to marginalize over all paths [Mohri et al., 2002]. For example, Mohri et al. [2002] construct a finite state transducer that combines the phone HMMs, the pronunciation dictionary and the tri-gram language model for a speech recognizer with 40,000 words and end up with more than 18 million states and 21 million transitions after compressing the state space with determinization and reduction. Thus, an approximate posterior is usually computed by marginalizing over a subset of all possible transcripts. Successful application of MMI for large vocabulary tasks requires that a lot of alternative hypotheses be considered, and as a result, the method only really flourished after recognizers started using lattice based methods [Valtchev et al., 1996, 1997, Woodland and Povey, 2000, Mohri et al., 2002]. Several refinements and alternatives have been proposed for discriminative training of GMM-HMM models - all with a goal of improving sequence level objectives [McDermott et al., 2007, Povey et al., 2008].

Sequential training methods have also been used with DNN-HMM systems [Kingsbury, 2009, Vesel et al., 2013, Jaitly et al., 2012, Su et al., 2013, Kingsbury et al., 2012]. These methods rely on computing differences of frame level posteriors from the correct transcripts and the frame level posteriors from all allowable transcripts. Thus learning requires decoding of each utterance, which can be a very time consuming proposition. For example Sainath et al. [2013] report that a fast sampling and preconditioning-based Hessian-free sequential discriminative training algorithm takes 44.5 hours to train on 50 hours of

Broadcast news data using 12 CPUs, where each CPU is an 8 core Intel Xeon X5570@2.93GHz.

Previous work has shown that training of GMM-HMMs with discriminative sequence-level objectives leads to improved phone accuracy even without improving the frame classification error [Gillick et al., 2012]. In this thesis all the models this far were trained to improve the log probability of the correct class (and implicitly, the classification error). It is logical to wonder whether an alternative objective function such as MMI would lead to improved results. However sequence training methods usually produce large gains only for LVCSR and gains for smaller databases such as TIMIT are typically smaller. In fact, in our experiments with TIMIT we only got small gains - for example for 6 layer network runs using FBANKs as input, the gains were around 0.2%. Mohamed et al. [2010] also report using a conditional random field (CRF) for full sequence training of a speech recognizer on TIMIT. The CRF uses a DNN trained with on forced alignment data to provide unary potentials of the CRF which is used to perform decoding. The DNN is fine-tuned with a sequence objective. This work reports a small gain for a two layer network. For a six layer network a gain of 0.6% was achieved only after very careful training by averaging over 800 utterances before each gradient update.

In this chapter we describe a simple but effective alternative method for training a neural network that uses partial sequential information [Jaitly and Hinton, 2014]. and produces strong gains in both phone recognition on TIMIT and large vocabulary speech recognition on WSJ. In this approach a Deep Neural Network (DNN) is trained to predict the forced-alignment state of multiple frames together using a separate softmax for each of the frames. This is in contrast to the usual method of training a DNN to predict only the state of the central frame. By itself this is not sufficient to improve accuracy of the system significantly. However, if we average the predictions for each frame - from the different contexts it is associated with - we achieve state of the art results on TIMIT using a fully connected Deep Neural Network without convolutional architectures or dropout training. On a 14 hour subset of Wall Street Journal (WSJ) using a context dependent DNN-HMM system it leads to a relative improvement of 6.4% on the dev set (*test-dev93*) and 9.3% on test set (*test-eval92*).

We then apply this method for training DNNs for speech recognition using the methods we developed in this thesis in chapters 3, 5 and 6.

7.1 Introduction

The use of forced alignments from Gaussian Mixture Model - Hidden Markov Models (GMM-HMM) for training neural networks suffers from several drawbacks. Firstly, the quality of the GMM-HMM system itself affects the quality of alignments generated. GMM-HMM systems make strong assumptions about the data generation process, such as independence of acoustic frames given states, that are not quite true. As a result the GMM-HMM model suffers from weird artifacts [Gillick et al., 2011]. In figure 7.1 we present results of an experiment that shows that forced alignments may not provide the best data to train neural networks with. For this, we used the forced alignments from a simple GMM-HMM system and corrupted the boundaries between phone internal states. To be more specific, we generated forced alignments from a tri-state monophone GMM-HMM system trained on TIMIT. For each segment corresponding to a phoneme we re-segmented the internal state boundaries by distributing them equally within the three internal states. Thus each segment between the start frame and the end frame assigned to a phoneme was split into three equal subsegments, and these were assigned the start state, the middle state and the end state of the triphone HMM. The effect of this is to generate an alignment that is

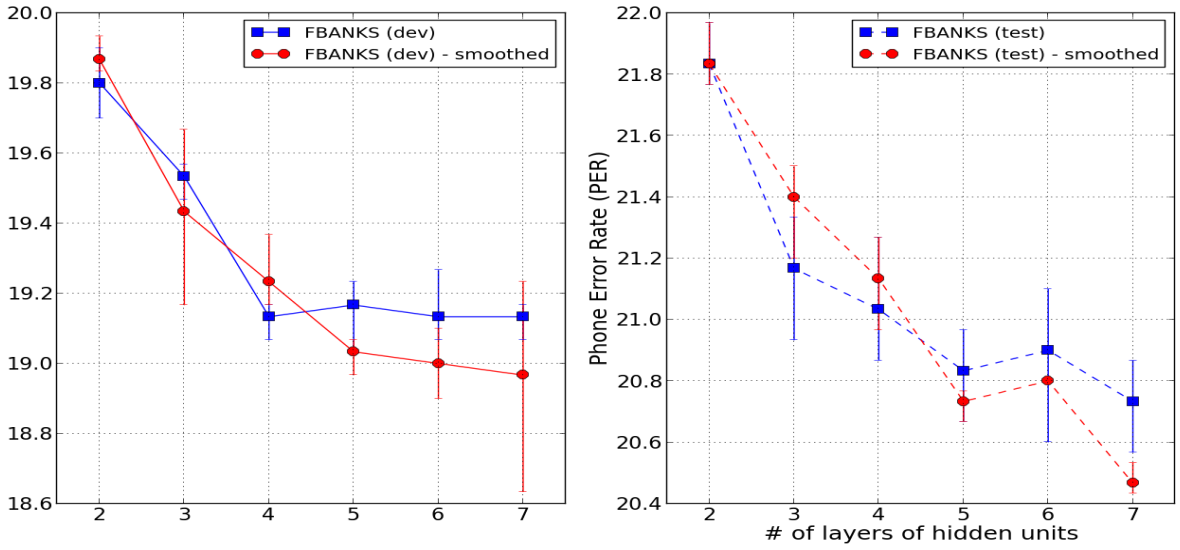


Figure 7.1: *Phone and Word Error rates (PER, WER) obtained with models trained on correct and smoothed alignments*

.

smoothed out. Note that the amount of data presented to the DNN models is the same in both cases - the procedure does not augment that training data because the smoothing is done only at the start of the training. So it should not lead to better DNN models through regularization. Figure 7.1 shows the results of training models of different depths using the smoothed alignments for TIMIT. Clearly, the phone recognition accuracy of the models trained with the corrupted boundaries is as good as, if not actually *better* than that the models trained with the forced alignments. The reason for these potential improvements is beyond the scope of this chapter, but might be explained by more robust estimates of rare states or better matching the equal transition probability of the HMM. It has been shown that discriminative training of GMM-HMMs leads to qualitatively different type of forced alignments that can lead to improved phone accuracy without improving the frame classification error [Gillick et al., 2012]. Gillick et al. [2012] used a criterion known as Minimum Phone Error [Povey and Woodland, 2002] and they suggest that this must mean that discriminative training's 'benefit only appears across sequences of frames'. It is natural to ask if neural networks trained even on forced alignments (not just in sequence discriminative training) could leverage some sequential information.

Another major drawback of the standard neural network training for ANN-HMM systems is that each data case only provides $\log_2 M$ bits of information through the state labels (where M is the number of distinct states). While this drawback is shared by all classification algorithms, speech is a very structured modality and this structure is ignored in the neural network training. Recurrent neural network methods trained on forced alignments (such as [Graves et al., 2013, Robinson, 1994, Robinson et al., 1996]) do not suffer from this problem since the entire sequence of targets is trained together and thus structured outputs are implicitly modelled by these methods.

In this chapter we present a method that attempts to incorporate these insights into neural network training from forced alignments. We train a neural network to predict the phone states of all the frames within a context window of a central frame using the acoustic data around the same central frame with

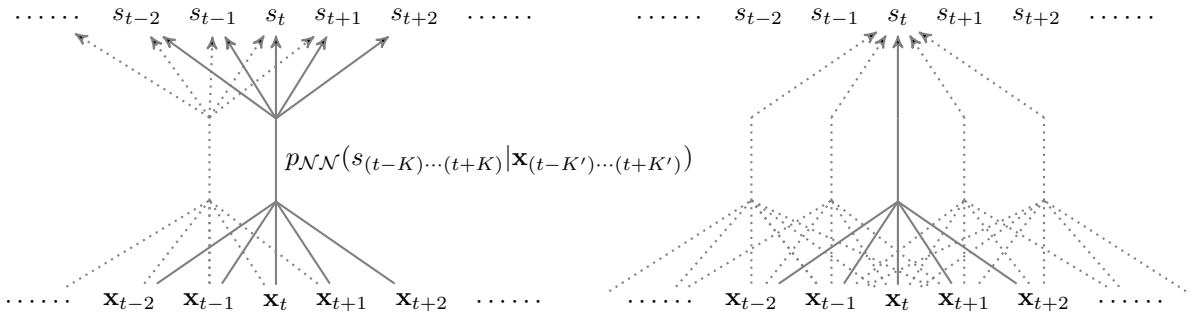


Figure 7.2: *Auto-regressive product model for speech recognition.* **Left:** *During training the neural network uses an input context of K' at each time point t to predict all the states within a context of K around t. In this example $K, K' = 2$.* **Right:** *During decoding the autoregressive scores for time point t from all contexts is averaged.*

the same (or larger) context window as input. At test time we take a geometric average (product model) of the predictions for each frame from all the acoustic contexts that model the state of that frame in their output layer. Using this method we achieve state of the art results on TIMIT using a vanilla fully connected DNN - without any special techniques such as Convolutional neural network architectures or dropout training Abdel-Hamid et al. [2013, 2012], Hinton et al. [2012b]. On a 14 hour subset of WSJ using a context dependent DNN-HMM system it leads to a relative improvement of 6.4% on the dev set (*test-dev93*) and 9.3% on test set (*test-eval92*).

Our approach to predicting multiple frames of outputs is hardly the first one. Recently Vanhoucke et al. [2013] trained a DNN to predict the output labels for multiple frames. However, no averaging of predictions was performed - instead the method was used to speed up decoding by performing forward passes through the DNNs at a slower frame rate. Product models are not new to the hybrid speech recognition community either. Sim [2009] shows an interesting application of product models to cross-lingual phone recognition by training a phone recognizer for English phones using a product model of ANN-HMM recognizers in Czech, Hungarian and Russian. Guangsen and Sim [2011] used a product of experts to predict context dependent state probabilities using predictions of a context independent model. Our approach is different from these two approaches in that we form a product from the same model applied convolutionally. In addition, at least for this chapter, we do not learn the weights of the distributions making up the product. We simply use the equally weighted geometric mean of the distributions.

7.2 Methods

Here we describe the training of the neural networks and the decoding. Figure 7.2 presents an overview of the method.

7.2.1 Multi-frame Target Training

We train a neural network to predict the phone labels for multiple output frames given the acoustic data. The inputs to the neural networks are $2K' + 1$ frames of acoustic vectors $\mathbf{x}_{t-K'} \cdots \mathbf{x}_{t+K'}$ and the targets are the $2K + 1$ one-hot encoded phone labels $\mathbf{s}_{t-K} \cdots \mathbf{s}_{t+K}$ associated with K context frames

around the center frame at time t[1]. The output layer is a set of $2K+1$ independent softmaxes, each with M phoneme label classes. Thus the conditional distribution of the output labels around time t is given by:

$$p\left(\mathbf{s}_{(t-K)\cdots(t+K)}|\mathbf{x}_{(t-K')\cdots(t+K')}\right) = \prod_{t'=t-K}^{t+K} p_{(t'-t)}(\mathbf{s}_{t'}|\mathbf{x}_{(t-K')\cdots(t+K')}) \tag{7.1}$$

where $p_{(t'-t)}(\mathbf{s}_{t'}|\mathbf{x}_{(t-K')\cdots(t+K')})$ is the softmax associated with a delay of $(t'-t)$ frames from the central frame.

The gradients with respect to the parameters of the neural network can be computed by backpropagating the the softmax gradients at the the output layer. Learning is done using stochastic gradient descent (more details can be found in section 7.3).

7.2.2 Decoding With Autoregressive Targets (DART)

Once a model been trained to perform multi-frame target prediction all the predictions for the states of an individual frame can be combined together. Specifically, we take a geometric average of all the predictions for each time point t:

$$p(\mathbf{s}_t|\mathbf{X}) \propto \prod_{t'=t-K}^{t+K} p_{t-t'}^{\frac{1}{2K+1}}(\mathbf{s}_t|\mathbf{x}_{(t'-K')\cdots(t'+K')}) \tag{7.2}$$

This is easily accomplished by averaging the activations of the softmaxes (i.e. the logit values) associated with time t from the application of the neural networks to all time points $t-K$ to $t+K$. Extra inputs are appended to the start and end of the utterances for boundary cases - we simply repeat the input at the first frame for prefix frames and the last frame for suffix frames. DART does not need to average over all K frames but experiments suggest that using all the frames leads to better results (see table 7.1).

Alternatively, instead of product averaging we could use a mean of all the predictions at each time point t, i.e.:

$$p(\mathbf{s}_t|\mathbf{X}) = \frac{1}{2K+1} \sum_{t'=t-K}^{t+K} p_{t-t'}(\mathbf{s}_t|\mathbf{x}_{(t'-K')\cdots(t'+K')}) \tag{7.3}$$

A comparison of these two averaging techniques is performed in section 7.3.5.

7.3 Experiments With FBANK Inputs

We used TIMIT [Garofolo et al., 1993] and the 14 hour subset of WSJ [Paul and Baker, 1992] (*si-84*) with FBANK inputs for exploring the various aspects of this method in this section. Experiments with the features discovered in this thesis are reported in section 7.4.

For the DNN-HMM models we used 15 frames of 40 dimensional filter banks with deltas and accelerations as inputs. All the neural networks using the same inputs were pretrained with the same Deep Belief Network [Hinton et al., 2006, Hinton and Salakhutdinov, 2006] - thus differences in results are not due to differences in initialization. We trained the deep neural networks using the annealing

[1] We use bold-face $\mathbf{s}_t$ for one-hot encoded labels and s_t for the class label that it is assigned to.

Table 7.1: *DART results on TIMIT from five layer DNNs trained on different output context window sizes. For each context window size we decoded using different window sizes of autoregressive output targets. In each case 15 frames of acoustic inputs (i.e. $K' = 7$) was used; each hidden layer had 2000 sigmoid units. Results are averages over three runs.*

training context (K)	**DART context**	**PER**	
		dev	**test**
0	0	19.2	20.8
1	0	18.9	20.6
	1	18.9	20.3
3	0	18.5	20.4
	1	18.3	20.3
	3	18.0	20.0
7	0	18.7	20.8
	1	18.5	20.5
	3	18.0	20.0
	7	17.6	19.5

schedule and learning rates described in chapter 2 with a small but critical change - the learning rates for the bottom two layers were reduced to 0.005 and 0.02 respectively. For TIMIT we used DNNs with several hidden layers of sigmoid units; for WSJ we used six layers as a compromise between depth and computational time. The TIMIT recipe had 180 output labels so the output layer was a set of $2K + 1$ softmaxes of 180 dimensions each - leading to output dimensions of $(2K+1)*180$. For WSJ we had 3385 states so the output layer was a set of $(2K+1)$ softmaxes of 3385 dimensions each. Because of the large output dimensions, training was slower. Note that even though we average multiple predictions there is only one DNN system, so the number of parameters is almost the same as the number of parameters in a vanilla DNN-HMM system. The only difference is the extra parameters at the output layer, where we have as many softmaxes as we have number of target frames, instead of a single softmax. When $K = 0$, i.e. we only predict the central state, these two are equivalent.

7.3.1 Effect of Context Window Sizes

Table 7.1 shows the Phone Error Rate (PER) achieved on TIMIT using DNNs with five hidden layers of sigmoid units trained to predict different context sizes of outputs. We trained four sets of triplicate models to predict the labels of 1, 3, 7 and 15 frames respectively (i.e. $K = 0, 1, 3, 7$) using 15 frames of input (i.e. $K' = 7$)[2]. For each model we averaged over different context window sizes to explore the impact of averaging. It can be seen that for each of these models averaging over multiple contexts improved the results. For example for the model with $K = 7$ the test set PER improved from 20.8% when no averaging was used (DART context = 0) to 19.5% when geometric averaging was performed over a context size of 7. Since averaging is constrained by the value of the training context, K, it makes sense to use the largest value possible. Of course it is a sensible choice to use $K \leq K'$. Thus we need to increase both these values for improved results. However we found no significant gains in improving K, K' beyond seven.

Interestingly, all models had comparable accuracy when they used only the central frame for predictions. This leads us to conclude that gains are due to model averaging rather than a regularization effect

[2] $K = 0$ is the baseline representing the vanilla DNN training for hybrid models.

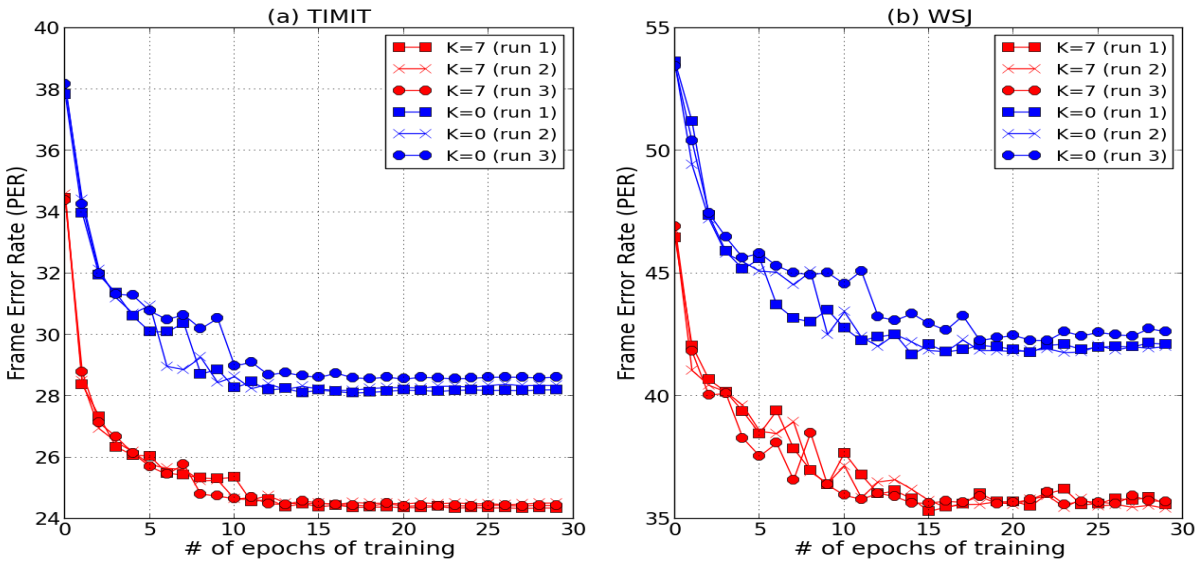


Figure 7.3: *Frame Error Rate (FER) of DART.* **Left:** *FER of DART on TIMIT using DNNs with five hidden layers of 2000 units each. Each model used +/-7 frames of context (i.e. $K' = 7$). The DNNs were trained to predict +/-7 frames of context for the multi-frame models ($K = 7$) and only the central frame for the non-multi-frame models ($K = 0$).* **Right:** *FER of DART on WSJ using DNNs with six hidden layers of 2000 units each. Each model used +/-7 frames of context. The DNNs were trained to predict +/-7 frames of context for the multi-frame models ($K = 7$) and only the central frame for the non-multi-frame models ($K = 0$).*

on the central frame prediction, achieved by predicting sub-states for more than the central frame.

7.3.2 Frame Error Rate of Models

We found that in addition to improved phone recognition results, the models also achieved much better frame error rates (FER) and log probabilities (which were computed using the geometrically averaged probability distributions). Figure 7.3 shows a comparison of the FER for three different runs of multi-frame DNN training (K=7) and the FER for three different runs of vanilla DNN training (K=0). Clearly, the multi-frame strategy leads to much better FER.

7.3.3 Impact of Depth of DNNs

Using $K, K' = 7$ we then trained three DNNs of depths two to seven hidden layers on TIMIT. Figure 7.4 shows the PER achieved for different depths and compares it to the baseline with $K = 0$. It can be seen that DART leads to a significant gain in accuracy over the baselines for both the dev set and the test set.

Note that we used fully connected deep neural network (DNN) models for this and achieved accuracy significantly better than those reported for simple Convolutional Neural Network - Deep Neural Network (CNN-DNN) - HMM systems [Abdel-Hamid et al., 2013, 2012] and comparable to carefully crafted CNN-DNN-HMM model with heterogeneous pooling in Deng et al. [2013] that was trained with dropout. It is our expectation that the gains are complementary, and similar gains would be produced when these

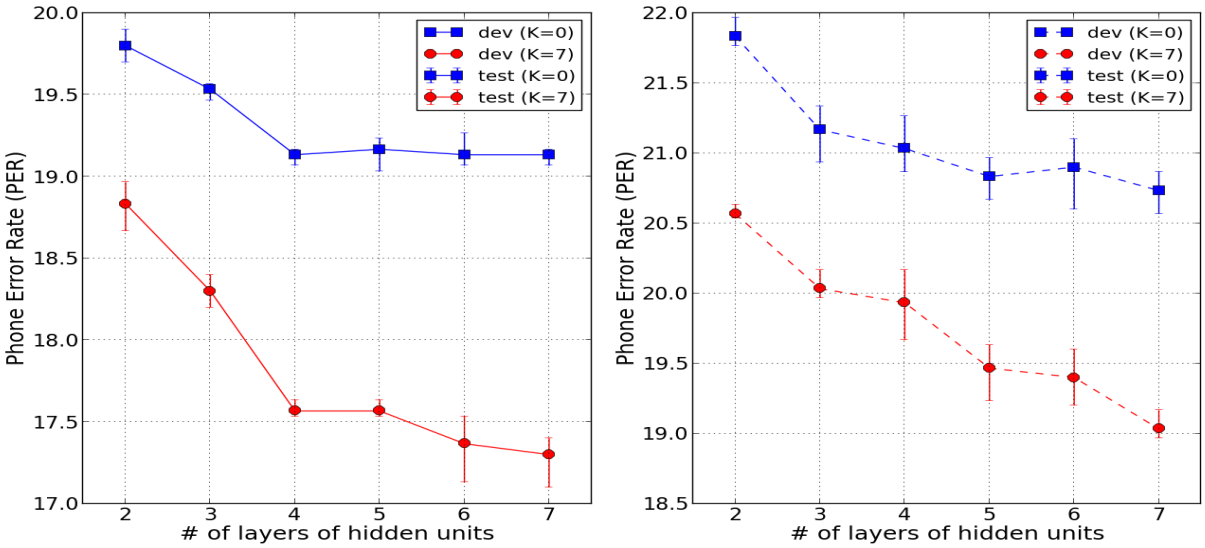


Figure 7.4: *PER of DART on TIMIT using DNNs of different depths. Each model used +/-7 acoustic frames of context ($K' = 7$). The DNNs were trained to predict +/-7 frames of context for the multi-frame models ($K = 7$) and only the central frame for the regular models ($K = 0$).*

Table 7.2: *DART results using context window of +/-7 in input data and output states. Results are averages over three runs.*

dataset	architecture	product	PER / WER	
			dev	test
TIMIT	$(2K)^7 - 180$	N	19.1	20.9
	$(2K)^7 - (180 \times 15)$	Y	**17.3**	**19.0**
WSJ-si84	$(2K)^6 - 3385$	N	9.4	5.4
	$(2K)^6 - (3385 \times 15)$	Y	**8.8**	**4.9**

ideas are applied to convolutional and other discriminative models.

7.3.4 WSJ Results

For WSJ, $K, K' = 7$ was used. Figure 7.3 shows the FER for triplicate runs of multi-frame target models and compares it to the FER for three runs of single target models. Clearly a much better FER is achieved once more.

Table 7.2 shows a numerical summary of the results. Since, $K = 7$ was used for these models, the output dimensionality was 3385x15. It can be seen that for *WSJ-si84* a relative improvement of 6.4% was seen on the development set *test-dev93* and a relative improvement of 9.3% was seen on the test set *test-eval92*, over the baseline system. Further improvements may be possible by experimenting with the parameters of the decoder but we did not explore this avenue.

Table 7.3: *A comparison of the geometric average to the arithmetic average of the autoregressive distributions on TIMIT test set using models of different depths. Results are averages over three runs.*

	# of layers of hidden units					
averaging type	**2**	**3**	**4**	**5**	**6**	**7**
geometric	20.6	20.0	19.9	19.5	19.4	19.0
arithmetic	20.8	20.3	20.4	19.9	19.6	19.4

7.3.5 Geometric Averaging Compared to Arithmetic Averaging

We decoded each of the models trained on TIMIT in section 7.3.3 using geometric and arithmetic averaging. Table 7.3 shows a comparison of the average PER over three runs. It is clear that geometric averaging consistently outperforms arithmetic averaging here. Also, the trend with depth is much more consistent for geometric averaging. A possible explanation for why geometric averaging outperforms arithmetic averaging is that geometric averaging acts like constraints - solutions that violate any one of the predictions sharply are discouraged under this model. Arithmetic averaging, on the other hand accepts solutions as long as one of the models is quite happy with the solution; thus it is susceptible to bad decision boundaries of models that have been over-fit significantly.

7.4 Experiments With Methods Developed in This thesis

We applied DART to each of the techniques from chapters 3, 5 and 6 [3]. We first discuss the results for TIMIT and then we discuss results on WSJ.

7.4.1 TIMIT

We trained DNNs using multi-frame targets for seven layer networks. For each of the techniques we used the DBNs that were trained to pretrain the DNNs in the previous chapters to pretrain the matching DNNs used here. We ran triplicate experiments for each category, with $K, K' = 7$ and for DART geometric averaging was performed using all +/-7 frames of context. Neural network training parameters were the same as those described above for the FBANK experiments in section 7.3.

Table 7.4 presents a summary of the results. It can be seen that significant, consistent gains were achieved by DART for each of the categories. For example, on the dev set, absolute gains of 2.2%, 1.8%, 1.3%, 1.7%, 1.6% and 1.7% were achieved on MFCC, FBANK, Gaussian-ReLU RBM features, frame-based templates, patch-based templates and gender specific VTLP respectively. Correspondingly for the test set, the absolute gains were 2.1%, 1.7%, 1.3%, 1.3%, 1.8% and 1.5% respectively. Using DART, the frame-based templates lead to no gains over FBANK features, but patch-based templates, and VTLP performed better than FBANKs, as they did without DART. In fact both these two methods outperform the best reported convolutional neural network performance on TIMIT [Deng et al., 2013] even without the use of dropout training [Hinton et al., 2012b]. Gaussian-ReLU RBM features seemed to gain the least from DART, but even so the improvement of 1.3% was very large. In fact with DART training MFCCs outperform Gaussian-Relu RBM features. It is uncertain whether this is due to the hyper-parameters involved in training the DNNs or some other factor - we did not attempt to tweak

[3] Note that VTLP could be combined with Capsules, but one would have to address the question of whether it is better to compute the instantiation parameters on the capsules on the perturbed or the unperturbed inputs of the capsules. For this thesis we did not address this model combination.

inputs	DART	dev	test
MFCC	N	20.6,20.4,20.4 (**20.5**)	22.6,21.9,22.1 (**22.2**)
	Y	18.5,18.2,18.2 (**18.3**)	20.3,20.1,20.0 (**20.1**)
FBANK	N	19.1,19.2,19.1 (**19.1**)	20.6,20.9,20.7 (**20.7**)
	Y	17.2,17.2,17.5 (**17.3**)	19.1,19.1,18.9 (**19.0**)
Gaussian-ReLU RBM features	N	20.5,20,6,20.5 (**20.5**)	22.5,22.1,22.4 (**22.3**)
	Y	19.2,19.1,19.3 (**19.2**)	21.0,20.9,20.9 (**21.0**)
FBANK + frame-based templates	N	19.1,19.0,18.8 (**19.0**)	20.5,20.1,20.4 (**20.3**)
	Y	17.3,17.1,17.2 (**17.2**)	18.9,18.9,18.9 (**18.9**)
FBANK + patch-based templates	N	18.6,18.5,18.4 (**18.5**)	20.6,20.5,20.4 (**20.5**)
	Y	16.8,16.9,17.0 (**16.9**)	18.7,19.0,18.5 (**18.7**)
FBANK + gender-specific VTLP	N	18.3,18.3,18.3 (**18.3**)	19.9,20.0,20.3 (**20.1**)
	Y	16.6,16.8,16.5 (**16.6**)	18.5,18.9,18.5 (**18.6**)

Table 7.4: Phone recognition results on TIMIT using DART with $K, K' = 7$. Inputs to the models were the features discovered in previous chapters.

the learning parameters for any of the method separately, choosing to use the same parameters for each DNN.

7.4.2 WSJ

For WSJ we used six layer DNNs. Again each DNN was pretrained with the same DBN used in previous chapters. Triplicate experiments were performed here as well. DART used $K, K' = 7$ and averaging was performed over all +/-7 frames of context.

Table 7.5 shows a summary of the results. The mean WER over the three experiments for each category are reported. In all cases, the FER improved significantly, but the WER does not show as consistent an improvement for WSJ as it did for TIMIT. For MFCC, FBANK and frame based templates, gender-specific VTLP gains of 0.5%, 0.6%, 0.4% and 0.2% respectively were seen on the dev set (test-dev93) and 0.4%, 0.5%, 0.4% and 0.2% on the test set (test-eval92). Path-based template showed no improvement over the dev set and an improvement of 0.2% on the test set, even though the best FER (=34.3%) on WSJ were observed for this case (compared to 39.6% in table 5.8 for patch based templates and 41.9% for FBANKs with training only on the center frame). Interestingly, patch-based templates did not perform very well on WSJ even for the DNN setup, even though, the FER was better than it was for FBANK by more than 2%. It is possible that tuning of decoding parameters could result in better accuracy - we did not try to explore this further.

7.5 Conclusions

We have shown that using an autoregressive product of a DNN-HMM system trained to predict the alignment labels of multiple frames can improve speech recognition accuracy. The autoregressive model bears a resemblance to RNNs because it attempts to predict states over a range of frames. These connections need to be further explored. In this paper the predictions at multiple time points were trained independently and a simple geometric average was used at test time. Model combination approaches frequently benefit by using weighted combinations. In the future we will explore these avenues further. It is interesting to note that geometric averaging outperforms arithmetic averaging here; it will be

inputs	DART	test-dev93	test-eval92
MFCC	N	9.7,9.7,9.9 (**9.8**)	6.0,5.8,5.6 (**5.8**)
	Y	9.2,9.6,9.0 (**9.3**)	5.4,5.5,5.2 (**5.4**)
FBANK	N	9.3,9.2,9.6 (**9.4**)	5.3,5.4,5.4 (**5.4**)
	Y	8.9,8.7,8.9 (**8.8**)	5.0,4.7,5.0 (**4.9**)
Gaussian-ReLU RBM features	N	10.7,10.6,10.7 (**10.7**)	5.8,5.6,5.8 (**5.7**)
	Y	10.7,11.0,10.9 (**10.9**)	5.4,5.5,5.7 (**5.5**)
FBANK + frame-based templates	N	9.0,9.0,9.1 (**9.0**)	5.1,5.3,5.4 (**5.3**)
	Y	8.6,8.6,8.6 (**8.6**)	5.0,4.9,4.9 (**4.9**)
FBANK + patch-based templates	N	9.1,9.1,9.3 (**9.2**)	5.4,5.4,5.6 (**5.5**)
	Y	9.2,9.2,9.2 (**9.2**)	5.3,5.3,5.3 (**5.3**)
FBANK + gender-specific VTLP	N	8.9,8.8,8.8 (**8.8**)	4.9,4.8,5.1 (**4.9**)
	Y	8.5,8.5,8.7 (**8.6**)	4.6,4.8,4.8 (**4.7**)

Table 7.5: Word recognition results on WSJ using DART with $K, K' = 7$. Inputs to the models were the features discovered in previous chapters.

interesting to see if this observation can be applied to training ensembles of models for speech recognition in new ways. We trained these models on forced alignments, but it is likely that sequential training methods (such as Kingsbury [2009]) could also benefit from this approach.

We note that gains made by features discovered in this thesis were not as pronounced with the application of DART on WSJ, even though big FER improvements were seen. It may be the the decoding parameters need further tuning - we did not attempt this because of the computational time involved in decoding. Nevertheless, we did note that the FERs improved significantly for each case.

Chapter 8

Conclusions

We conclude the thesis by summarizing our contributions and analyzing the errors made on TIMIT by each the methods in the previous chapters. Finally, we propose future research directions.

8.1 Summary of Contributions

This thesis had three main contributions related to Deep Learning and speech recognition.

Firstly, it developed unsupervised Deep Learning methods for discovering features from speech signals that were processed to different degrees. While unsupervised learning has been an important area for Deep Learning its application to speech recognition has previously been limited to using vanilla DBNs and autoencoders to pretrain neural networks that use Mel-log filter banks as inputs. In contrast we developed new models that used various frame-based representations of speech signals. The earlier methods were based on using little domain knowledge: chapter 3 used a Gaussian ReLU RBM to model raw speech signal. By learning standard deviations of the Gaussian units we could learn features from raw speech that resembled the response patterns of auditory nerve fibers. It was clear from these earlier explorations that domain knowledge was required to achieve better speech recognition accruacy and subsequent chapters in this part of the thesis attempted to do that. Chapter 4 showed how capsules could be learned from raw speech and from spectrograms using transforming autoencoders. Chapter 5 developed a new way of discovering capsules using an autoencoder with deformable templates and a domain specific decoder. It was applied to discover capsules in individual frames of spectrograms and patches of FBANK. The results from these exploration showed us that gains could indeed by achieved on TIMIT by augmenting traditional features with these features. Frame error rate improvements were achieved on WSJ but did not lead to gains in word error rates. It is possible that optimizing with sequence level objectives might have helped.

Secondly, in chapter 6 we developed a method for perturbing FBANK spectra to augment databases for speech recognition and showed that these perturbations lead to improved speech recognition accuracy. Such methods have previously been applied to object detection in vision to learn invariances to translations, small transformations etc, but to our knowledge have never been applied successfully to speech recognition.

Lastly, in chapter 7 we developed a new way of training and using DNNs for speech recognition that is motivated by sequence discriminative training methods, but uses a much simpler method. Instead of

training on computationally expensive sequence discriminative objectives, or recurrent neural networks trained on full utterances, we develop a method that models the targets over shorter segments of speech. This method was applied to train the DNNs used in all the earlier chapters and showed significant improvements in almost all cases. We believe that this opens up new avenues for training DNN-HMM systems.

8.2 Analysis of Errors

Table 8.1 shows the percentage errors for different types of broad phonetic classes for each of the algorithms using the triplicate seven layer DNNs trained in previous chapters. The rows are sorted by increasing error rate using FBANK inputs. It can be seen that for FBANK features vowels are the most difficult phonetic class (error 25.9%) whereas fricatives are the easiest (error 15.7%). This is not surprising. The variation in these sounds from person to person is much smaller than it is for vowels and other types of sound, because only turbulent flow at the tip of the mouth is involved in producing the sound, not the entire vocal apparatus. Vowel sounds on the other hand are influenced by the shape of the entire vocal cavity and thus have much greater variation from person to person. In addition they are more influenced by the context of the phone preceding and succeeding them. The same trend is also true for the other feature types and VTLP. For example for VTLP the error rate for vowels is 25.2% and for fricatives it is 15.3%.

Table 8.2 shows the actual counts for errors. For each category type we have emphasized the method that performs best for the class by using **bold face font**. Unsurprisingly, VTLP improves upon FBANK results through improvements in nasals, semi-vowels and glides and vowels - phonetic classes that involve the entire vocal tract. The accuracy for the same classes also improves with templates on frames. Templates on patches also improve over FBANK on these categories, except for Semi-vowels where they perform slightly worse (169 errors instead of 166). In addition both templates based methods show better accuracy on fricatives. This is presumably because they can learn to model the precise frequencies in spectrograms and patches of the spectrograms where high energy is distributed - the distinguishing characteristics of fricatives.

Table A.1 shows the breakdown for different individual phonemes.

Table 8.1: *Summary of the phone recognition error rates for different phone classes on the TIMIT dataset for each of the methods of chapters 3, 5 and 6.*

Type	MFCC	FBANK	GRRBM	templates (frames)	templates (patches)	VTLP
Pause	8.6	**7.6**	8.3	7.9	8.5	8.0
Fricatives	16.8	15.7	17.1	**15.3**	**14.8**	**15.3**
Stops	18.0	**16.3**	18.1	16.5	16.9	16.5
Nasals	21.8	19.2	22.3	**18.4**	**18.7**	**18.1**
Semi-vowels	22.1	20.0	23.0	**19.7**	20.3	**19.2**
Affricatives	38.1	23.8	33.3	23.8	23.8	23.8
Vowels	26.9	25.9	28.9	**25.3**	**25.2**	**25.3**

Table 8.3 shows the breakdown of errors for phonetic classes when DART was used for recognition, and table 8.4 shows the same errors as percentages. Compared to tables 8.2 and table 8.1 we see that DART produces gains in all categories for all the methods. The general trend of errors is the same as

Table 8.2: *Summary of the phone recognition errors for different phone classes on the TIMIT dataset for each of the methods of chapters 3, 5 and 6.*

Type	MFCC	FBANK	GRRBM	templates (frames)	templates (patches)	VTLP	# of instances
Insertions	197	199	**175**	**186**	**187**	**172**	N/A
Pause	136	**120**	131	124	134	127	1579
Fricatives	163	152	166	**148**	**143**	**148**	969
Stops	160	**145**	161	147	150	147	889
Nasals	140	123	143	**118**	**120**	**116**	641
Semi-vowels	184	166	191	**164**	169	**160**	832
Affricatives	16	10	14	10	10	10	82
Vowels	630	606	676	**593**	**589**	**593**	2341
Total	1626	1521	1657	**1490**	**1502**	**1473**	7333

it was before with a small, surprising difference - VTLP no longer performs better on vowels than do FBANK features. Instead the gains for VTLP over FBANKs seem to arise from fewer insertions and fewer errors with fricatives and stops. It is unclear why this is the case.

Table 8.3: *Summary of the phone recognition errors for different phone classes on the TIMIT dataset for each of the methods of chapters 3, 5 and 6 using DART.*

Type	MFCC	FBANK	GRRBM	templates (frames)	templates (patches)	VTLP	# of instances
Insertions	183	181	**156**	**176**	190	**162**	NA
Pause	126	115	127	115	**112**	117	1579
Fricatives	159	143	160	144	143	**137**	969
Stops	138	130	146	**129**	130	**124**	889
Nasals	121	105	132	106	**101**	**102**	641
Semi-vowels & Glides	159	148	166	**147**	**144**	**145**	832
Affricatives	12	9	14	**8**	10	9	82
Vowels	579	564	638	**559**	**541**	570	2341
Total	1477	1395	1539	**1384**	**1371**	**1366**	7333

8.3 Future Avenues for Exploration

In this paper feature learning was performed without regard for supervised training. The features learned were however applied to a supervised training problem - speech recognition. It is possible that semi-supervised learning of features - with supervision signals coming from phone-recognition information - would result in features that perform better on the speech recognition task. The speech recognition community has a large body of work showing that preprocessing of acoustic data with methods that enhance phonetic differences, such as Linear Discriminant Analysis result in better speech recognition. This could be done even for the Deep Learning methods by adding terms to the reconstruction error objective that encourage class separation. Deep Learning methods such as Non-Linear Components Analysis have been applied to such problems [Salakhutdinov and Hinton, 2007]. However it would be important to scale these methods to large problems since they rely on computing pairwise statistics between different points.

Another important property that features should possess is invariance to speakers. Our methods

Table 8.4: *Summary of the phone recognition error rates for different phone classes on the TIMIT dataset for each of the methods of chapters 3, 5 and 6 using DART.*

Type	MFCC	FBANK	GRRBM	templates (frames)	templates (patches)	VTLP
Pause	8.0	7.3	8.0	7.3	**7.1**	7.4
Fricatives	16.4	14.8	16.5	14.9	14.8	**14.1**
Stops	15.5	14.6	16.4	**14.5**	14.6	**13.9**
Nasals	18.9	16.4	20.6	16.5	**15.8**	**15.9**
Semi-vowels	19.1	17.8	20.0	**17.7**	**17.3**	**17.4**
Affricatives	14.6	11.0	17.1	**9.8**	12.2	11.0
Vowels	24.7	24.1	27.3	**23.9**	**23.1**	24.3

made no attempt to encourage this invariance explicitly either. Future attempts at feature discovery should address this very important problem because speaker adaptation often results in large gains in accuracy of speech recognition systems. One way to achieve this would be to add regularization terms to the objective function that encourage disentangling of speaker information from phoneme classes. For example encouraging a subset of features in the code layer to predict identity, and others to predict phoneme classes.

A promising approach that may achieve these objective is to use our ideas from chapter 5 with an articulatory model for speech as a decoder. We would couple a neural network encoder to a speech synthesizer that works with articulatory information and learn a code that is natural for speech - the configuration of the articulators. Changes in resting shape of the articulatory configurations would be used to model speakers themselves. Augmenting traditional spectral features with articulatory configurations has been shown to improve speech recognition accuracy [Zlokarnik, 1993, Wrench and Richmond, 2000, Metze, 2007]. Intuitively, this makes sense because articulatory configurations seem more consistent than the acoustic signals. For example the /s/ sound is typically produced when the tongue touches the palate and turbulent flow is generated at the constriction. This articulatory gesture is fairly well preserved from person to person and from utterance to utterance but the acoustic signals produced are highly variable. Articulatory configurations also reveal co-articulatory intentions during speech production - for example the rounding of the lips during the production of the sound /b/ in the word boot occurs because of the intention to produce the /u/ sound after /b/ - and these factors could be properly accounted for during recognition.

Of course articulatory inversion methods using neural networks have been tried in the past [Schroeter and Sondhi, 1994] to discover articulatory configurations. These lost favor partly because the one-to-many mapping from acoustic signal to articulatory configurations was perceived as too big a hurdle. However the recent availability of Electromagnetic Articulography data (e.g. [Rudzicz et al., 2012, Richmond et al., 2011]) provides information that could be used to constrain such inversion models so that the articulatory codes remain in reasonable parts of the space.

Appendix A

Breakdown of Phone Error Rates

Table A.1: *Summary of the phone recognition error rates for different phones on the TIMIT dataset for each of the methods of chapters 3, 5 and 6.*

Class	Phone	MFCC	FBANK	Gaussian-RRBM	templates(frames)	templates(patches)	VTLP
	pau	8.6	7.6	8.3	7.9	8.5	8.0
FRICATIVES	s	8.2	7.5	8.8	7.7	7.2	8.3
	sh	16.0	15.6	13.0	14.7	16.5	14.3
	z	22.1	21.5	20.3	23.2	19.2	21.2
	f	9.7	10.4	10.9	9.4	9.9	10.2
	th	41.2	28.9	35.1	28.9	28.9	26.3
	v	24.7	22.2	29.7	23.7	22.9	22.6
	dh	26.0	24.5	27.6	19.5	21.4	21.6
STOPS	b	14.4	13.1	14.9	17.2	15.7	15.7
	d	35.7	31.3	36.9	33.0	31.3	30.4
	g	16.7	17.2	18.2	13.6	15.2	16.2
	p	20.6	15.2	16.7	14.5	16.2	15.4
	t	15.9	14.6	16.2	14.4	16.0	15.2
	k	11.5	13.5	12.3	11.3	12.5	12.5
	dx	14.1	12.2	15.6	13.7	13.0	13.0
NASL	m	23.7	19.6	25.5	18.8	19.9	19.4
	n	19.0	18.0	19.4	16.8	17.3	15.9
	ng	35.9	26.9	30.8	28.2	25.0	28.2
SV & GL	hh	28.1	21.6	29.0	20.3	21.2	16.5
	l	20.5	19.9	20.6	18.9	19.0	17.8
	r	23.0	19.1	22.7	20.1	20.4	20.6
	w	15.7	15.5	16.9	13.7	17.4	15.0
	y	36.7	34.0	46.7	38.0	34.7	35.3
AFF	jh	17.5	11.1	14.3	12.7	9.5	9.5
	ch	22.5	14.2	19.2	12.5	14.2	14.2
VOWELS	aa	21.3	19.3	21.6	20.2	19.0	18.4
	ae	40.0	39.4	41.9	36.5	41.0	37.5
	ah	34.3	32.6	36.3	32.9	31.0	29.8
	aw	32.2	25.6	30.0	24.4	31.1	22.2
	ay	17.6	15.0	16.5	13.1	17.2	15.4
	eh	38.8	36.0	40.4	34.2	33.2	36.7
	er	21.5	24.6	24.4	22.5	20.9	22.6
	ey	16.4	12.3	15.5	12.9	13.7	13.5
	ih	24.3	24.6	25.6	23.7	23.6	25.2
	iy	17.3	16.2	18.2	16.0	15.5	15.6
	ow	25.1	26.2	35.6	25.5	26.6	27.0
	oy	43.8	47.9	54.2	56.2	45.8	33.3
	uh	79.3	73.6	73.6	74.7	71.3	73.6
	uw	35.1	28.4	49.5	30.2	32.9	30.6

Bibliography

O. Abdel-Hamid, A. 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*, pages 4277–4280. IEEE, 2012.

O. Abdel-Hamid, L. Deng, and D. Yu. Exploring convolutional Neural Network structures and optimization techniques for speech recognition. In *INTERSPEECH*, pages 3366–3370, 2013.

M. Abe, S. Nakamura, K. Shikano, and H. Kuwabara. Voice conversion through vector quantization. In *Acoustics, Speech, and Signal Processing, 1988. ICASSP-88., 1988 International Conference on*, pages 655–658. IEEE, 1988.

D. Albesano, R. Gemello, and F. Mana. Hybrid HMM-NN modeling of stationarytransitional units for continuous speech recognition. *Information Sciences*, 123(12):3 – 11, 2000.

L. Bahl, P. Brown, P. De Souza, and R. 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.*, volume 11, pages 49–52, Apr 1986.

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

L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov Chains. *The Annals of Mathematical Statistics*, 41(1): 164–171, 02 1970.

Y. Bengio, R. De Mori, G. Flammia, and R. Kompe. Global optimization of a neural network-hidden Markov model hybrid. *Neural Networks, IEEE Transactions on*, 3(2):252–259, 1992.

L. L. Beranek, L. L. Beranek, L. L. Beranek, and L. L. Beranek. *Acoustical Measurements*. Acoustical Society of America, 1988.

C. M. Bishop and N. M. Nasrabadi. *Pattern recognition and machine learning*, volume 1. springer New York, 2006.

L. Bottou. Online algorithms and stochastic approximations. In D. Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998. URL `http://leon.bottou.org/papers/bottou-98x`. revised, oct 2012.

H. Bourlard, N. Morgan, C. Wooters, and S. Renals. CDNN: A context dependent Neural Network for continuous speech recognition. In *Acoustics, Speech, and Signal Processing, 1992. ICASSP-92., 1992 IEEE International Conference on*, volume 2, pages 349–352. IEEE, 1992.

H. A. Bourlard and N. Morgan. *Connectionist Speech Recognition: A Hybrid Approach*. Kluwer Academic Publishers, Norwell, MA, USA, 1993. ISBN 0792393961.

L. H. Carney and T. Yin. Temporal coding of resonances by low-frequency auditory nerve fibers: single-fiber responses and a population model. *Journal of Neurophysiology*, 60(5):1653–1677, 1988.

D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber. High-performance Neural Networks for Visual Object Classification. *CoRR*, abs/1102.0183, 2011.

M. Cohen, H. Franco, N. Morgan, D. Rumelhart, and V. Abrash. Multiple-state context-dependent phonetic modeling with MLP. In *Proceedings of Speech Research Symposium XII*. Citeseer, 1992.

C. H. Coker and O. Fujimura. Model for specification of the vocal-tract area function. *The Journal of the Acoustical Society of America*, 40(5):1271–1271, 1966.

G. Dahl, D. Yu, L. Deng, and A. Acero. Context-dependent pre-trained Deep Neural Networks for large vocabulary speech recognition. *Audio, Speech, and Language Processing, IEEE Transactions on*, (99): 1–1, 2010.

S. Davis and P. Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 28(4):357–366, Aug. 1980.

A. P. Dempster, Laird, N. M., and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255, June 2009.

L. Deng, O. Abdel-Hamid, and D. Yu. A deep convolutional Neural Network using heterogeneous pooling for trading acoustic invariance with phonetic confusion. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6669–6673. IEEE, 2013.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

D. Ellis and N. Morgan. Size matters: An empirical study of Neural Network training for large vocabulary continuous speech recognition. In *Acoustics, Speech, and Signal Processing, 1999. Proceedings., 1999 IEEE International Conference on*, volume 2, pages 1013–1016. IEEE, 1999.

C. G. M. Fant. *Analysis and synthesis of speech processes*. North-Holland Publishing Comp., 1967.

J. L. Flanagan. *Speech Analysis, Synthesis and Perception*. Springer-Verlag, New York, 1972.

J. L. Flanagan and L. Cherry. Excitation of vocal-tract synthesizers. *The Journal of the Acoustical Society of America*, 45(3):764–769, 1969.

J. L. Flanagan, C. H. Coker, L. R. Rabiner, R. W. Schafer, and N. Umeda. Synthetic voices for computers. *Spectrum, IEEE*, 7(10):22–45, Oct. 1970.

H. Fletcher. Auditory patterns. *Reviews of Modern Physics*, 12(1):47, 1940.

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*, 8(3):211 – 222, 1994.

M. J. Gales. Maximum likelihood linear transformations for HMM-based speech recognition. *Computer speech & language*, 12(2):75–98, 1998.

M. J. Gales. Semi-tied covariance matrices for Hidden Markov Models. *Speech and Audio Processing, IEEE Transactions on*, 7(3):272–281, 1999.

J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, and N. L. Dahlgren. DARPA TIMIT acoustic phonetic continuous speech corpus CDROM, 1993. URL `http://www.ldc.upenn.edu/Catalog/LDC93S1.html`.

S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):721–741, 1984.

D. Gillick, L. Gillick, and S. Wegmann. Don't multiply lightly: Quantifying problems with the acoustic model assumptions in speech recognition. In *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*, pages 71–76, Dec 2011.

D. Gillick, S. Wegmann, and L. Gillick. Discriminative training for speech recognition is compensating for statistical dependence in the HMM framework. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 4745–4748, March 2012.

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*, pages 532–535. IEEE, 1989.

J. R. Glass. A probabilistic framework for segment-based speech recognition. *Computer Speech & Language*, 17(2):137–152, 2003.

A. Graves, N. Jaitly, and A. Mohamed. Hybrid speech recognition with Deep Bidirectional LSTM. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pages 273–278, Dec 2013.

S. Greenberg, C. D. Geisler, and L. Deng. Frequency selectivity of single cochlear-nerve fibers based on the temporal response pattern to two-tone signals. *The Journal of the Acoustical Society of America*, 79:1010, 1986.

W. Guangsen and K. C. Sim. Sequential classification criteria for NNs in automatic speech recognition. In *INTERSPEECH*, 2011.

E. Helander, H. Silen, T. Virtanen, and M. Gabbouj. Voice conversion using dynamic kernel partial least squares regression. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(3):806–817, March 2012.

H. Hermansky. Perceptual linear predictive (PLP) analysis of speech. *The Journal of the Acoustical Society of America*, 87(4):1738–1752, 1990.

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. *Signal Processing Magazine*, 2012a.

G. E. Hinton. Training products of experts by minimizing Contrastive Divergence. *Neural Computation*, 14(8):1711–1800, 2002.

G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with Neural Networks. *Science*, 313(5786):504–507, July 2006.

G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for Deep Belief Nets. *Neural Computation*, 18(7):1527–1554, 2006.

G. E. Hinton, A. Krizhevsky, and S. Wang. Transforming auto-encoders. In *ICANN-11: International Conference on Artificial Neural Networks*, 2011.

G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving Neural Networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012b.

K. Ishizaka and J. Flanagan. Synthesis of voiced sounds from a two-mass model of the vocal cords. *Bell Syst. Tech. J.*, 51:1233–1268, 1972.

N. Jaitly and G. Hinton. Using an autoencoder with deformable templates to discover features for automated speech recognition. In *INTERSPEECH*, 2013a.

N. Jaitly and G. E. Hinton. Learning a better representation of speech soundwaves using Restricted Boltzmann Machines. In *ICASSP*, pages 5884–5887, 2011a.

N. Jaitly and G. E. Hinton. A new way to learn acoustic events. In *Advances in Neural Information Processing Systems 24, Deep Learning workshop*. MIT Press, 2011b.

N. Jaitly and G. E. Hinton. Vocal Tract Length Perturbation (VTLP) improves speech recognition. In *International Conference on Machine Learning (ICML)*, 2013b.

N. Jaitly and G. E. Hinton. Autoregressive product of multi-frame predictions can improve the accuracy of hybrid models. In *INTERSPEECH*, pages –, 2014.

N. Jaitly, P. Nguyen, A. W. Senior, and V. Vanhoucke. Application of pretrained Deep Neural Networks to large vocabulary speech recognition. In *INTERSPEECH*, 2012.

D. Jurafsky, J. H. Martin, A. Kehler, K. Vander Linden, and N. Ward. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, volume 2. MIT Press, 2000.

A. Kain and M. W. Macon. Spectral voice conversion for text-to-speech synthesis. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, volume 1, pages 285–288. IEEE, 1998.

N. Kanda, R. Takeda, and Y. Obuchi. Elastic spectral distortion for low resource speech recognition with deep neural networks. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pages 309–314, Dec 2013.

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*, pages 3761–3764. IEEE, 2009.

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.

W. Koenig. A new frequency scale for acoustic measurements. *Bell Telephone Laboratory Record*, 27: 299–301, 1949.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1106–1114, 2012.

Y. LeCun and C. Cortes. The MNIST database of handwritten digits. `http://yann.lecun.com/exdb/mnist/`, 1998.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

J.-H. Lee, H.-Y. Jung, T.-W. Lee, and S.-Y. Lee. Speech feature extraction using Independent Component Analysis. In *Acoustics, Speech, and Signal Processing, 2000. ICASSP'00. Proceedings. 2000 IEEE International Conference on*, volume 3, pages 1631–1634. IEEE, 2000.

L. Lee and R. Rose. A frequency warping approach to speaker normalization. *Speech and Audio Processing, IEEE Transactions on*, 6(1):49–60, Jan 1998.

C. J. Leggetter and P. Woodland. Maximum likelihood linear regression for speaker adaptation of continuous density hidden markov models. *Computer Speech & Language*, 9(2):171–185, 1995.

M. S. Lewicki. Efficient coding of natural sounds. *Nature Neuroscience*, 5(4):356–363, 2002.

M. S. Lewicki. Information theory: A signal take on speech. *Nature*, 466(7308):821–822, 2010.

J. C. Licklider. Basic correlates of the auditory stimulus. In S. S. Stevens, editor, *Handbook of experimental psychology*. Wiley, Oxford, England, 1951.

J. S. Liu, W. H. Wong, and A. Kong. Covariance structure of the Gibbs sampler with applications to the comparisons of estimators and augmentation schemes. *Biometrika*, 81(1):27–40, 1994.

J. Makhoul and L. Cosell. LPCW: An LPC vocoder with linear predictive spectral warping. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'76.*, volume 1, pages 466–469. IEEE, 1976.

E. McDermott, T. J. Hazen, J. Le Roux, A. Nakamura, and S. Katagiri. Discriminative training for large-vocabulary speech recognition using Minimum Classification Error. *Audio, Speech, and Language Processing, IEEE Transactions on*, 15(1):203–223, 2007.

R. Memisevic and G. Hinton. Unsupervised learning of image transformations. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.

F. Metze. Discriminative speaker adaptation using articulatory features. *Speech Communication*, 49(5): 348–360, 2007.

T. Mikolov, A. Deoras, S. Kombrink, L. Burget, and J. Cernockỳ. Empirical evaluation and combination of advanced language modeling techniques. In *INTERSPEECH*, pages 605–608, 2011.

A. Mohamed, D. Yu, and L. Deng. Investigation of full-sequence training of Deep Belief Networks for speech recognition. In *INTERSPEECH*, pages 2846–2849, 2010.

A. Mohamed, G. E. Dahl, and G. Hinton. Acoustic Modeling using Deep Belief Networks. *Trans. Audio, Speech and Lang. Proc.*, 20(1):14–22, Jan. 2012. ISSN 1558-7916.

M. Mohri, F. Pereira, and M. Riley. Weighted Finite-State Transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88, 2002.

B. C. Moore and B. R. Glasberg. Suggested formulae for calculating auditory-filter bandwidths and excitation patterns. *The Journal of the Acoustical Society of America*, 74(3):750–753, 1983.

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*, pages 413–416. IEEE, 1990.

N. Morgan and H. Bourlard. Continuous speech recognition. *Signal Processing Magazine, IEEE*, 12(3): 24–42, 1995.

V. Nair and G. E. Hinton. Rectified linear units improve Restricted Boltzmann Machines. In *Proc. 27th International Conference on Machine Learning*, 2010.

H. Ney and S. Ortmanns. Dynamic programming search for continuous speech recognition. *Signal Processing Magazine, IEEE*, 16(5):64–83, 1999.

D. O'Shaughnessy. Linear Predictive Coding. *Potentials, IEEE*, 7(1):29–32, 1988.

M. Padmanabhan, G. Saon, and G. Zweig. Lattice-based unsupervised MLLR for speaker adaptation. In *ASR2000-Automatic Speech Recognition: Challenges for the new Millenium ISCA Tutorial and Research Workshop (ITRW)*, 2000.

D. Palaz, R. Collobert, and M. Magimai-Doss. Estimating phoneme class conditional probabilities from raw speech signal using convolutional neural networks. In *INTERSPEECH*, pages 1766–1770, 2013.

R. D. Patterson. Auditory filter shapes derived with noise stimuli. *The Journal of the Acoustical Society of America*, 59(3):640–654, 1976.

D. B. Paul and J. M. Baker. The design for the Wall Street Journal-based CSR corpus. In *DARPA Speech and Language Workshop*. Morgan Kaufmann Publishers, 1992.

J. W. Picone. Signal modeling techniques in speech recognition. *Proceedings of the IEEE*, 81(9):1215–1247, 1993.

A. Poritz. Linear predictive Hidden Markov Models and the speech signal. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP '82.*, volume 7, pages 1291–1294, May 1982.

D. Povey. *Discriminative training for large vocabulary speech recognition.* PhD thesis, Cambridge University, 2004.

D. Povey and P. Woodland. Minimum Phone Error and I-smoothing for improved discriminative training. In *Acoustics, Speech, and Signal Processing (ICASSP), 2002 IEEE International Conference on*, volume 1, pages I–105–I–108, May 2002.

D. Povey, D. Kanevsky, B. Kingsbury, B. Ramabhadran, G. Saon, and K. Visweswariah. Boosted MMI for model and feature-space discriminative training. In *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, pages 4057–4060. IEEE, 2008.

D. Povey, L. Burget, M. Agarwal, P. Akyazi, F. Kai, A. Ghoshal, O. Glembek, N. Goel, M. Karafit, A. Rastrow, R. C. Rose, P. Schwarz, and S. Thomas. The subspace Gaussian Mixture Model a structured model for speech recognition. *Computer Speech & Language*, 25(2):404 – 439, 2011a.

D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer, and K. Vesely. The Kaldi Speech Recognition Toolkit. In *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding.* IEEE Signal Processing Society, Dec. 2011b.

L. R. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, Feb. 1989.

L. R. Rabiner and R. W. Schafer. *Digital Processing of Speech Signals*, volume 19. IET, 1979.

M. Ranzato and G. E. Hinton. Modeling pixel means and covariances using factorized third-order boltzmann machines. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2551–2558. IEEE, 2010.

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*, 2(1):161–174, 1994.

K. Richmond, P. Hoole, and S. King. Announcing the electromagnetic articulography (day 1) subset of the mngu0 articulatory corpus. In *INTERSPEECH*, pages 1505–1508, 2011.

A. Robinson. An application of recurrent nets to phone probability estimation. *Neural Networks, IEEE Transactions on*, 5(2):298–305, Mar 1994.

T. Robinson, M. Hochberg, and S. Renals. The use of recurrent Neural Networks in continuous speech recognition. In *Automatic Speech and Speaker Recognition*, pages 233–258. Springer, 1996.

S. Roth and M. J. Black. Fields of experts: A framework for learning image priors. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 860–867. IEEE, 2005.

F. Rudzicz, A. Namasivayam, and T. Wolff. The TORGO database of acoustic and articulatory speech from speakers with dysarthria. *Language Resources and Evaluation*, 46(4):523–541, 2012. ISSN 1574-020X.

T. N. Sainath, L. Horesh, B. Kingsbury, A. Y. Aravkin, and B. Ramabhadran. Accelerating Hessian-free optimization for deep neural networks by implicit preconditioning and sampling. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pages 303–308. IEEE, 2013.

R. Salakhutdinov and G. E. Hinton. Learning a nonlinear embedding by preserving class neighbourhood structure. In *International Conference on Artificial Intelligence and Statistics*, pages 412–419, 2007.

R. Salakhutdinov, S. Roweis, and Z. Ghahramani. Optimization with EM and Expectation-Conjugate-Gradient. In *ICML*, pages 672–679, 2003.

G. Saon, S. Dharanipragada, and D. Povey. Feature space gaussianization. In *Acoustics, Speech, and Signal Processing, 2004. Proceedings.(ICASSP'04). IEEE International Conference on*, volume 1, pages I–329. IEEE, 2004.

J. Schalkwyk, D. Beeferman, F. Beaufays, B. Byrne, C. Chelba, M. Cohen, M. Kamvar, and B. Strope. your word is my command: Google search by voice: A case study. In *Advances in Speech Recognition*, pages 61–90. Springer, 2010.

J. Schroeter and M. M. Sondhi. Techniques for estimating vocal-tract shapes from the speech signal. *Speech and Audio Processing, IEEE Transactions on*, 2(1):133 –150, jan 1994.

H. Sheikhzadeh and L. Deng. Waveform-based speech recognition using hidden filter models: parameter selection and sensitivity to power normalization. *Speech and Audio Processing, IEEE Transactions on*, 2(1):80–89, Jan 1994.

K. C. Sim. Discriminative product-of-expert acoustic mapping for cross-lingual phone recognition. In *Automatic Speech Recognition & Understanding, 2009. ASRU 2009. IEEE Workshop on*, pages 546–551. IEEE, 2009.

P. Simard, D. Steinkraus, and J. Platt. Best practices for convolutional Neural Networks applied to visual document analysis. In *Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on*, pages 958–963, 2003.

E. C. Smith and M. S. Lewicki. Efficient auditory coding. *Nature*, 439(7079):978–982, 2006.

P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. In D. E. Rumelhart, J. L. McClelland, et al., editors, *Parallel Distributed Processing: Volume 1: Foundations*, pages 194–281. MIT Press, Cambridge, 1986.

H. Soltau, G. Saon, and B. Kingsbury. The IBM Attila speech recognition toolkit. In *Spoken Language Technology Workshop (SLT), 2010 IEEE*, pages 97–102. IEEE, 2010.

F. K. Soong and E.-F. Huang. A tree-rellis based fast search for finding the n-best sentence hypotheses in continuous speech recognition. In *Acoustics, Speech, and Signal Processing, 1991. ICASSP-91., 1991 International Conference on*, pages 705–708. IEEE, 1991.

K. N. Stevens, S. Kasowski, and C. G. M. Fant. An electrical analog of the vocal tract. *The Journal of the Acoustical Society of America*, 25(4):734 – 742, 1953.

S. S. Stevens, J. Volkmann, and E. B. Newman. A scale for the measurement of the psychological magnitude pitch. *The Journal of the Acoustical Society of America*, 8(3):185–190, 1937.

Y. Stylianou, O. Cappé, and E. Moulines. Continuous probabilistic transform for voice conversion. *Speech and Audio Processing, IEEE Transactions on*, 6(2):131–142, 1998.

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*, pages 6664–6668, 2013.

J. Susskind, A. Anderson, and G. Hinton. The Toronto Face Database. Technical Report 001, Department of Computer Science, University of Toronto, 2010.

G. W. Taylor, G. E. Hinton, and S. T. Roweis. Modeling human motion using binary latent variables.

T. Tieleman. *Optimizing Neural Networks that Generate Images*. PhD thesis, University of Toronto, Toronto, Canada, June 2014.

T. Toda, A. W. Black, and K. Tokuda. Voice conversion based on maximum-likelihood estimation of spectral parameter trajectory. *Audio, Speech, and Language Processing, IEEE Transactions on*, 15 (8):2222–2235, 2007.

E. Trentin and M. Gori. A survey of hybrid ANN/HMM models for automatic speech recognition. *Neurocomputing*, 37(14):91 – 126, 2001.

V. Valtchev. *Discriminative methods in HMM-based speech recognition*. PhD thesis, University of Cambridge, 1995.

V. Valtchev, J. Odell, P. Woodland, and S. Young. Lattice-based discriminative training for large vocabulary speech recognition. In *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on*, volume 2, pages 605–608. IEEE, 1996.

V. Valtchev, J. Odell, P. C. Woodland, and S. J. Young. MMIE training of large vocabulary recognition systems. *Speech Communication*, 22(4):303–314, 1997.

V. Vanhoucke, M. Devin, and G. Heigold. Multiframe Deep Neural Networks for acoustic modeling. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 7582–7585, May 2013.

K. Vesel, A. Ghoshal, L. Burget, and D. Povey. *Sequence-discriminative training of deep neural networks*. ISCA, 2013.

W. Walker, P. Lamere, P. Kwok, B. Raj, R. Singh, E. Gouvea, P. Wolf, and J. Woelfel. Sphinx-4: A flexible open source framework for speech recognition. Technical report, Sun Microsystems, Inc., Mountain View, CA, USA, 2004.

S. Wegmann, D. McAllaster, J. Orloff, and B. Peskin. Speaker normalization on conversational telephone speech. In *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on*, volume 1, pages 339–341. IEEE, 1996.

M. Welling, M. Rosen-Zvi, and G. E. Hinton. Exponential family harmoniums with an application to information retrieval. In *Advances in neural information processing systems*, pages 1481–1488, 2004.

P. Woodland and D. Povey. Large scale discriminative training for speech recognition. In *ASR2000-Automatic Speech Recognition: Challenges for the new Millenium ISCA Tutorial and Research Workshop (ITRW)*, 2000.

A. Wrench and K. Richmond. Continuous speech recognition using articulatory data. In *Proc. ICSLP 2000*, Beijing, China, 2000.

H. Ye and S. Young. Perceptually weighted linear transformations for voice conversion. In *INTERSPEECH*, 2003.

S. Young. Statistical Modeling in Continuous Speech Recognition (CSR). In J. S. Breese and D. Koller, editors, *UAI*, pages 562–571. Morgan Kaufmann, 2001. ISBN 1-55860-800-1.

S. Young, G. Evermann, D. Kershaw, G. Moore, J. Odell, D. Ollason, V. Valtchev, and P. Woodland. The HTK book. *Cambridge University Engineering Department*, 3:175, 2002.

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

I. Zlokarnik. Experiments with an articulatory speech recognizer. In *EUROSPEECH'93*, pages –1–1, 1993.