LEARNING HIERARCHICAL SPEECH REPRESENTATIONS USING DEEP CONVOLUTIONAL NEURAL NETWORKS

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF MASTER OF PHILOSOPHY
IN THE FACULTY OF ENGINEERING AND PHYSICAL SCIENCES

2014

By
Darren Hau
School of Computer Science

Contents

Word Count: 16244

List of Tables

List of Figures

Abstract

LEARNING HIERARCHICAL SPEECH REPRESENTATIONS
USING DEEP CONVOLUTIONAL NEURAL NETWORKS
Darren Hau
A thesis submitted to the University of Manchester
for the degree of Master of Philosophy, 2014

Deep learning has proven to be an effective methodology in handling complex AI problems, especially for visual perception tasks. Key to the success of deep learning is its ability to learn hierarchical feature representations of increasing levels of abstraction. Motivated by the success of deep learning in the visual domain, researchers have recently begun to apply deep learning to speech. In this study, we are interested in investigating the feasibility of using deep convolutional neural networks (CNN) in the speech domain. CNNs were designed based on models of the visual system and have been shown to learn hierarchical feature representations on vision tasks. As many vision tasks have an auditory analogue, we believe deep CNNs could learn an effective hierarchical representation for speech. In the speech domain, most deep architectures have used a Restricted Boltzmann Machine (RBM) based deep architecture. A secondary aim of this study is to determine whether or not a different building block can be used effectively in the speech domain. We construct a deep architecture using the CNN as the building block trained using unsupervised learning only. We compare our work against a Convolutional RBM based model on various speech perception tasks showing that it is indeed possible to use an alternative to the RBM in the speech domain. Our analysis also leads to some non-trivial observations on the suitability of using CNN-based deep architectures in the speech domain.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

I would like to thank my supervisor, Dr. Ke Chen, for his support throughout this work. I'd also like to thank my colleagues in the Machine Learning and Optimisation Group, my friends and my family for helping me through the difficulties that I have faced. Finally, I'd like to thank the EPSRC for funding me during this work.

Chapter 1

Introduction

1.1 Motivation

Speech signals are a complex, temporal data source containing a mixture of different information components ranging from linguistic information, speaker-specific information and environmental information. While it is effortless for the human auditory system to extract the relevant information from the speech signal for a specific speech perception task, it is a major challenge for automatic speech information processing systems. For example, in an automatic speech recognition system, only the linguistic information is required whilst the speaker information is not. The reverse is true for a speaker recognition system whilst in both systems, any environmental information serves only to confuse and harm the performance of the system. Extracting a suitable speech representation for a specific task with only the relevant information components required is still an open problem.

The most commonly used representations in automatic speech information processing systems are based on signal processing methods. These representations, such as Mel-Frequency Cepstral Coefficients (MFCC), may also take into account of some of the basic characteristics of the human auditory system. MFCCs and similar representations are compact and efficient to compute. As such, they have remained popular for the past couple of decades.

However, in order to achieve a compact representation, MFCCs and other similar representations, throw away a lot of the information contained within a speech signal - information that could be very useful for certain perceptual tasks. We believe that using the deep learning strategy in the speech domain can provide an alternative speech representation that addresses the above issues.

Deep learning provides a novel methodology for dealing with complex AI problems. Its successes are mainly attributed to hierarchical unsupervised learning that leads to effective hierarchical feature representations of increasing levels of abstraction [33] [15]. To perform hierarchical unsupervised learning, Hinton et al. proposed a systematic strategy that trains the building blocks of a deep architecture in a greedy layer-wise manner [33] with the output of each layer forming the hierarchical representation.

Originally, deep learning was applied mostly to visual perception tasks with many reported successes [34] [85] [25]. Motivated by this success, deep learning has expanded into the domain of speech perception. Prior to when this work was originally carried out at the end of 2011, only a few works had appeared applying deep learning to speech. This included [55] for learning a generic audio representation used in various audio classification tasks, [11] for learning a generic speaker-specific representation and [59] that used a deep neural network as the acoustic model in a phoneme recognition system. The then state-of-art-results achieved in [59] resulted in a new wave of research into using deep learning for speech recognition and continually advanced the state-of-the-art in that field [32].

Returning to 2011, we noted that the works in [59] and [55] use the Restricted Boltzmann Machine (RBM) [33] as the building block to create a deep architecture, specifically a deep belief network. Studies on deep learning strategies [5] [49] suggested that a simple multi-layer perceptron (MLP) with one hidden layer trained as an autoencoder and using the same deep learning strategy proposed by Hinton et al. [33], may be an alternative to using the RBM in constructing deep architectures. In particular, a deep architecture constructed using a variant of the autoencoder, the denoising autoencoder [88], often outperforms an RBM-based counterpart in OCR and object recognition tasks [15].

The convolutional neural network (CNN) architecture is a biologically inspired variant of an MLP originally developed for visual perception tasks and trained using supervised learning [51]. An unsupervised learning algorithm, named Predictive Sparse Decomposition (PSD) [43], was proposed to enable the creation of a CNN trained using the deep learning strategy [33]. Using a multi-layered CNN trained using unsupervised learning leads to hierarchical representations that turn out to be crucial in achieving state-of-the-art performance in various visual perception tasks [40]. There are many similarities between the visual cortex and auditory cortex in the brain with many visual tasks having an auditory analogue [78]. We believe that it may be possible

to use the CNN to create hierarchical speech representations much like those achieved in the vision domain.

1.2 Aims and Objectives

The aim of this work is to carry out a feasibility study into the use of deep convolutional neural networks trained on speech signals. Here we define "deep" as meaning trained using a methodology similar to the deep learning strategy [33]. In this instance, we use greedy, layer-wise unsupervised training to learn the parameters of a convolutional neural network in order to learn what we hope to be a good representation of speech.

More specifically, we are interested in determining whether a CNN building block can be used as an alternative to the RBM-based building blocks used in [59] and [55]. In order to do so, we will follow the work in [55] adopting the same corpus (TIMIT [18]) and experimental settings in order to make our study completely comparable.

1.3 Contributions

The contributions of this study are as follows. We believe we are the first to apply CNNs and the PSD learning algorithm on the speech domain. Comparative results on speech perception tasks including gender classification, speaker identification and phoneme classification, as well as a qualitative analysis of the learned features demonstrates that an unsupervised CNN can be an alternative to the RBM-based architecture in the domain of speech.

In comparison to the baseline performance with MFCCs, we also observe that the convolutional architecture, originally designed for visual perception, might not be sufficient for speech information processing as required by various speech perception tasks.

This work was originally published at the 11th UK Workshop on Computational Intelligence, 2011 [26].

1.4 Thesis Outline

This thesis is structured as follows. In Chapter 2, we review essential background concepts such as speech production, spectrograms, MFCCs and deep learning. In Chapter

3, we describe the CNN model and training algorithm used for this study. In Chapter 4, we present the results of the comparison study against [55] and discuss their significance in Chapter 5. Finally, we conclude in Chapter 6.

Chapter 2

Background

In this chapter, we give a basic overview of the human speech production and speech processing systems. We then turn our attention to how speech can be represented, including conventional signal processing techniques such as the spectrogram and Mel-Frequency Cepstral Coefficients. In addition, we review the emerging technique of deep learning which has proven very successful in learning hierarchical feature representations.

2.1 Speech Production and Speech Processing

Speech production begins with the speaker formulating a message in their brain. The brain then co-ordinates the movement of the vocal organs to produce the necessary sounds that represents the message in the speaker's chosen language. An illustration of the human vocal system is shown in Fig. 2.1.

Speech sounds are a result of the motion of air passing through the vocal system. Puffs of air are drawn up from the lungs, passing through the larynx, which is more commonly known as the voice box. The larynx houses the vocal folds, or the glottis, which then vibrate as the air passes through. The vibration of the vocal folds occur at a particular base frequency and is known as the fundamental frequency which is the major determinant of the pitch of a speaker's voice.

Some sounds however do not require the vibration of the vocal folds and these sounds are known as unvoiced sounds. Examples of unvoiced sounds are [p] and [s]. For voiced sounds, the rapid opening and closing (vibration) of the vocal folds leads to periodic pulses of sound whilst unvoiced sounds result in a white noise like sound.

After the air passes through the larynx, it enters the vocal tract consisting of the oral

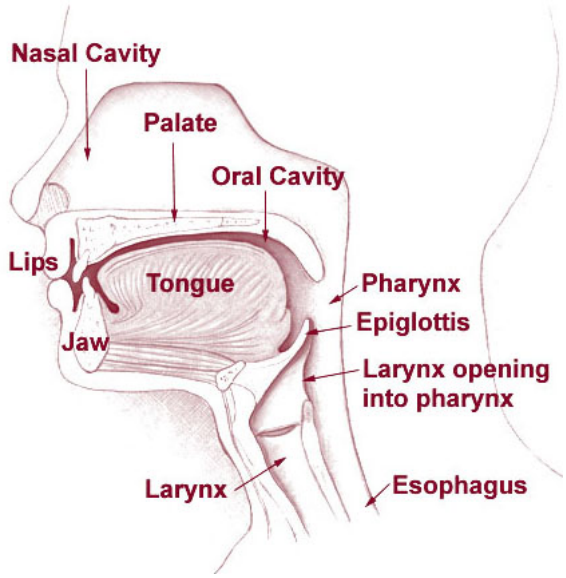


Figure 2.1: Illustration of the vocal system [89]

tract and nasal tract. Sounds then exit through the mouth and/or nose. Consonants are the result of the restriction of the airflow at specific positions in the vocal tract whilst for vowels, there is less restriction. Vowel sounds are dependent on the shape of the vocal tract such as the tongue and the lips. The restrictions and the shape of the vocal tract create resonances known as formant frequencies or formants which can help to identify the speech sound being produced.

The vocal tract and folds are unique for each person and this gives rise to a speaker's characteristic voice which can be used as a means of identification. This leads to variations in the speech sounds which must be compensated for during the processing of speech by a listener. In addition, the speaker's emotional state, whether they are happy, sad or angry can result in vastly different speech output.

Before a speech signal reaches a listener, the signal must be transmitted through some medium from the speaker to the listener. This might be through the air or down telephone lines but no matter what the medium is, the speech signal will be distorted by the medium of transmission and these are known as channel and environmental factors. These factors must also be compensated for by a listener.

The processes by which this compensation occurs and the decoding of the speech signal itself back into the original intended message is not well understood. What is known however is that sound waves are converted to neural signals by the peripheral auditory system.

Sound waves are captured by the outer ear and directed into the auditory canal

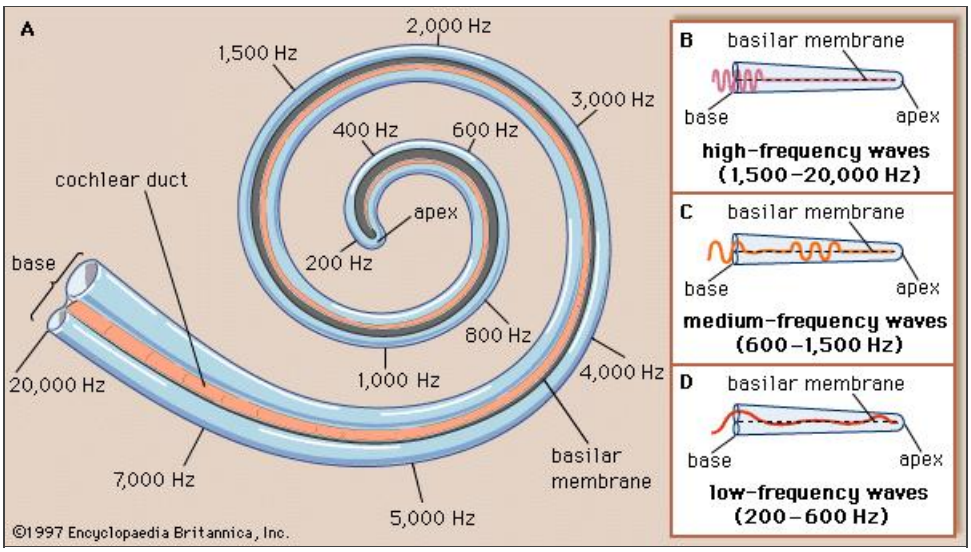


Figure 2.2: Illustration of the frequency sensitivity of the basilar membrane [63]

which causes the eardrum to vibrate. These vibrations cause movement in the fluids inside the spiral shaped cochlea of the inner ear. The cochlea is split along its length by the basilar membrane. The movement of the fluid causes the displacement of the basilar membrane which is sensitive to different frequencies of sound as shown in Fig. 2.2. In this way, the cochlea is able to perform a frequency analysis of the incoming sound. The frequency scale is non-linear and frequency resolution is higher for low frequencies than for high frequencies. This frequency based or tonotopic organization is maintained throughout the auditory system.

Hair cells transcode the motion of the basilar membrane into neural firing rates in the auditory nerve for processing in the auditory cortex. The auditory cortex is responsible for speech information processing tasks, for example, decoding the linguistic message or the speaker identity from the neural firing patterns.

2.2 Digital Signal Processing Based Speech Representations

We now turn our attention to the way in which speech is represented in automated speech processing systems. The most common methods are grounded in digital signal processing and are based around the frequency or spectral analysis of a digitised speech signal. Here, we will review two methods used in this work, the spectrogram and Mel-Frequency Cepstral Coefficients.

2.2.1 Spectrogram

Spectrograms are a 2D-plot showing the frequency composition of a signal over time. A signal is divided up into short time segments using a window function such as the Hamming window. It is assumed that the signal is stationary within each window and so a short time Fourier analysis can be performed on each segment. The computed magnitude spectrum at each time frame can be collected together into a plot of time vs. frequency with energy levels shown as colour intensity. Often the energy levels or intensity is shown on a logarithmic scale. An example spectrogram for the sentence, "she had your dark suit in greasy wash water all year", from the TIMIT corpus [18] is shown in Fig. 2.3.

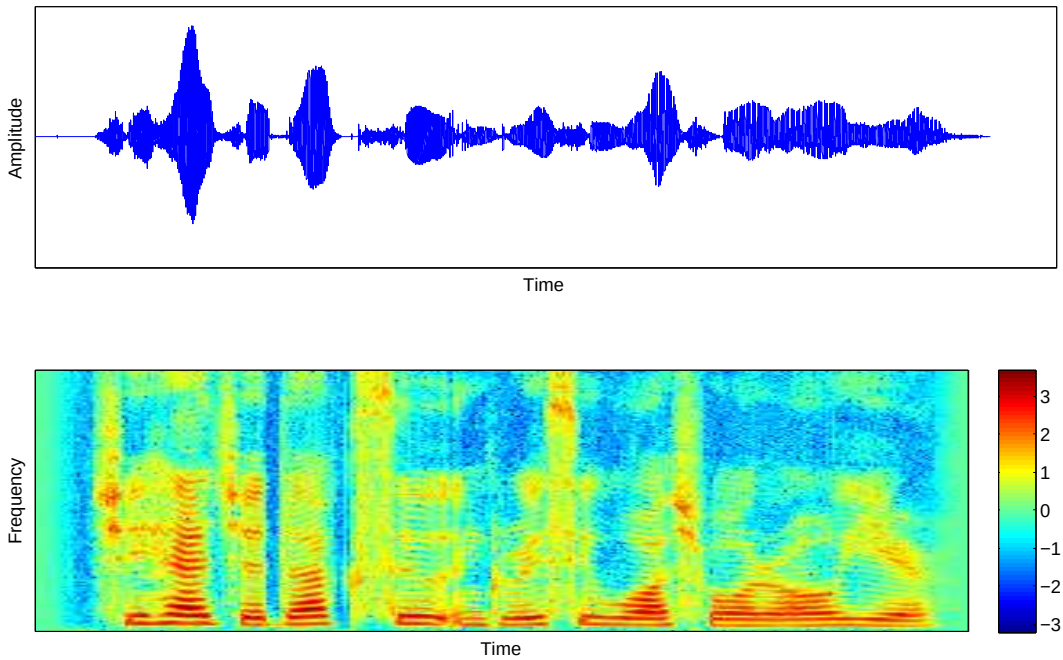


Figure 2.3: Speech signal (top) and corresponding spectrogram (bottom) for the sentence, "she had your dark suit in greasy wash water all year", from the TIMIT corpus [18].

The spectrogram can be analysed to detect different types of speech sound, extract the pitch, the formant frequencies and other acoustic features. However, there is a trade-off between the frequency resolution and temporal resolution of the spectrogram. This trade-off is controlled by the duration of the window function, with high frequency resolution requiring window durations of approximately 20 - 30ms, whilst

high temporal resolution requires windows with durations of 10ms or less. High frequency resolution spectrograms are known as narrow-band spectrograms whilst high temporal resolution spectrograms are known as wide-band spectrograms. Narrow-band spectrograms allow the harmonic frequencies (the human voice is a harmonic sound) to be visible whilst wide-band spectrograms show the periodicities in the speech signal.

The spectrogram itself is not usually directly used as a speech representation due to its high dimensionality. Often, a dimensionality reduction technique such as Principal Component Analysis (see Sec. 2.3) is applied to make computations more feasible. Other lower dimensional speech representations such as Mel-Frequency Cepstral Coefficients (MFCC) tend to be more commonly used however.

An alternative to spectrograms are cochleagrams which are computed using a model of the cochlea such as the two models described in [80]. These models try to mimic the functionality of the cochlea, in particular, the frequency selectivity and motion of the basilar membrane in response to the detection of sound waves, the non-linear detection and conversion of basilar membrane motion into auditory nerve firing rates by the inner hair cells and the continuous adaptation of the system to changing activity and sound input levels.

Compared to a spectrogram where there is a trade-off between frequency resolution and temporal resolution, the cochleagram is able to retain fine time structure and uses a biologically plausible frequency resolution. The cochleagram is able to show details such as the glottal pulses, harmonics and formant tracks.

Cochleagrams tend to be used in in the field of Computational Auditory Scene Analysis (CASA) where the goal is to segment a sound signal according to the different sound sources or objects present [35] where precise timing and frequency details can be especially useful. In the majority of other speech information processing tasks, for example, speech recognition and speaker identification, spectrograms and MFCC-like representations have proven more popular than cochleagrams.

2.2.2 Mel-Frequency Cepstral Coefficients

Mel-Frequency Cepstral Coefficients (MFCC) are a popular low dimensional representation of the speech signal. The process of computing MFCCs is shown in the block diagram in Fig. 2.4. MFCCs are calculated by first taking the short-time Fourier transform of a windowed segment of the speech signal, usually 20 - 40ms in duration. The frequency axis of the spectrum is mapped onto the Mel scale. The Mel scale is a perceptually adjusted frequency scale that attempts to emulate the non-linear frequency

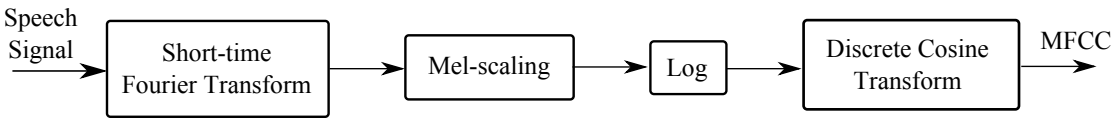


Figure 2.4: Block diagram showing the processes involved in computing MFCC representation

analysis of the cochlea. The Mel scale is approximately linear up to 1000Hz and increases logarithmically thereafter. This transformation is performed using overlapping triangular windows that try to emulate the critical band frequencies of the cochlea.

The logarithm of the mel-spectrum is then performed in order to reflect the non-linear response to changes in sound level by the auditory system. This log-spectrum is further analysed using either a second Fourier transform or the related discrete cosine transform (DCT) resulting in what is known as the mel-cepstrum.

To establish why this is useful, we can consider speech production as a source-filter model, whereby the source is the glottal pulse waveform produced by the vocal folds vibrating at a particular fundamental frequency and the filter is the shape of vocal tract with its own resonant frequencies. For speech recognition, we are only interested in the filter, the shape of the vocal tract which leads to a particular speech sound being produced.

The DCT is useful as it can separate the speech signal into its source and filter components, with the lower coefficients of the mel-cepstrum corresponding to the filter components and higher coefficients to the source components. The DCT also has excellent energy compaction properties meaning only as few as 12 cepstral values are required for representing the filter characteristics for speech recognition. For speaker recognition, we are also interested in the source components and the fundamental frequency of the speech signal. In practice, 19 cepstral values are used for speaker related tasks. As speech has a temporal dynamic component, MFCCs are often augmented with their first and second derivatives known as delta and delta-delta values.

Traditional automated speech recognition systems use Hidden Markov Models (HMM) combined with Gaussian Mixture Models (GMM) for modelling the likelihood of each HMM state generating a particular acoustic feature vector. MFCCs are ideal for use as acoustic feature vectors with GMMs as the DCT used in computing MFCCs is an orthogonal transform. This means that cepstral coefficients are decorrelated and therefore GMMs with diagonal covariance matrices can be used instead of GMMs with full covariance matrices. Diagonal covariance matrix GMMs have fewer

parameters and therefore require less training data and less computation time to calculate the likelihood. This combined with the low dimensionality of MFCCs and the efficient computation of the DCT has seen the enduring popularity of the MFCC representation since the 1980s when they were first introduced.

Examples of automatic speech recognition systems that use this approach include [37] [57] [90]. Additionally, MFCC feature vectors with GMM modelling have been used successfully in speaker identification and verification tasks [45] [67]. MFCCs have also been used with techniques other than GMM/HMM including augmented conditional random fields [29].

MFCCs however are not a very robust representation and are susceptible to noise factors from channel and environmental conditions often requiring some form of noise reduction [86] [66] [93]. MFCCs can be described as low level acoustic features derived directly from the speech signal. However, humans are known to use multiple levels of processing and just as with analysing individual pixel intensities for visual tasks, MFCCs may be too low-level to achieve optimal performance.

2.3 Principal Component Analysis

Principal Component Analysis (PCA) can be used to find the statistical distribution of a dataset. PCA is an orthogonal transformation of the dataset such that the basis vectors span the directions of maximum variance of the data. As such, PCA can be used for dimensionality reduction, with the top N components chosen so that a certain percentage of the variance of the data is retained. PCA being an orthogonal transformation also has the effect of decorrelating the data which makes it a useful pre-processing step for machine learning algorithms.

In the context of speech applications, as mentioned previously, PCA can be used for dimensionality reduction of spectrograms and it is this representation that forms the input features for our work. As shown in Fig. 2.5, the process of computing PCA transformed spectrograms is very similar to the process of computing MFCCs (Fig. 2.4).

PCA, also known as the Karhunen-Loéve Transform (KLT), is very similar to the Discrete Cosine Transform (DCT) used for MFCCs. Both are orthogonal transforms capable of decorrelating data. When the PCA transform is computed from coherent time series data such as speech, PCA finds basis functions similar to the cosine basis functions used in the DCT [81] [12]. This is no surprise as one of the original

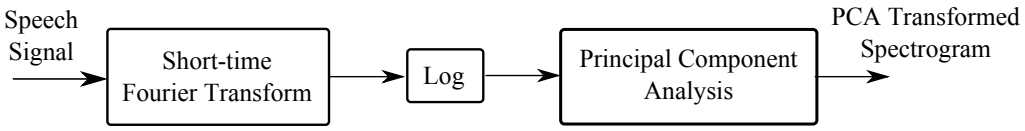


Figure 2.5: Block diagram showing the processes involved in computing PCA transformed spectrogram

motivations of the DCT was to find a method that could compute the KLT quickly [3].

PCA however, is a data-driven method and there is no guarantee that it can produce the source-filter separation that the DCT is capable of for MFCCs. In our work, we compute the PCA transform on the spectrogram of a whole utterance. This naturally results in quite a different representation to MFCCs as the DCT is performed on short windows of speech. Being data driven can be an advantage though as it can lead to more suitable basis functions being found.

However, PCA being a linear transform, it is unlikely that the basis functions found will correspond to the information components of speech due to the complex and non-linear relationship between information components. In particular, as the dominant information component in speech is the linguistic information, PCA will likely throw away relatively minor information components such as the speaker information during the dimensionality reduction process. This of course would be disastrous for speaker recognition tasks.

2.4 Sparse Coding

Sparse coding supposes that neural representations of natural stimuli should reflect the statistics of natural data, maximizing the amount of information whilst also minimizing redundancy in the representation. What this means in practice is that in general for a particular stimulus, only a small number of neurons will be active out of a large population.

Using these principles, Olshausen and Field [62] presented an algorithm for learning a sparse representation of natural images. An input, x, is represented as a linear combination of a set of basis functions, ϕ, each weighted by the coefficients, a_i, as shown in Eq. 2.1:

$$x = \sum_{i=1}^{n} a_i \phi_i \tag{2.1}$$

Following the sparse coding principle, the number of non-zero coefficients will be few in number with the majority of the coefficients being zero. This means that an input can be described by a linear combination of a small number of basis functions. The actual set of basis functions used to explain the dataset is large, in some cases over-complete, meaning that the number of basis functions is greater than the dimensionality of the input itself. This allows for many different possible variations in the input data.

The basis functions are learned unsupervised on a dataset by minimizing the mean squared error between the reconstructed representation and the actual input, constrained by a sparsity parameter such as the L_1-norm. In [62], the learned basis functions, trained using natural images, were similar to the receptive fields of simple cells in the visual system.

In automatic speech recognition, sparse coding has been used in conjunction with exemplar-based methods. In exemplar-based sparse representations, the basis functions, or exemplars, are selected from the training set. Each data instance is then represented by a small linear combination of these training exemplars. This new feature space can then be used for training an HMM based speech recogniser. For test data, this remapping helps to move the test data distribution closer to that of the training set distribution helping to improve recognition [73]. This led to a reduction from 19.9% to 19.2% phoneme error rate on the TIMIT dataset compared to a conventional GMM/HMM system.

In [20], exemplar-based sparse representations were used for missing data imputation. Missing data imputation seeks to replace noisy parts of speech features with clean speech estimates. Speech features are labelled as reliable (clean) or unreliable (noisy) and sparse coding is used to find a sparse linear combination of clean speech exemplars from the training set that minimizes the reconstruction error of the reliable speech features. The clean speech exemplars serve as estimates for the unreliable speech features. The new sparse representation can be fed into a speech recognition system alongside uncertainty measurements for the estimated clean speech replacements which helps in the decoding process. This results in a substantial improvement in recognition accuracy over conventional missing data techniques [19] [20].

Away from exemplar-based methods, [79] uses a sparse autoassociator to learn both the basis functions and sparse encoding for speech data. A sparse autoassociator is a neural network with one hidden layer trained using a sparse coding objective function (see Sec. 2.5.2 for more details on autoassociators, also known as autoencoders).

This new learned sparse representation is used as the feature representation for a hybrid neural network/HMM speech recogniser. Compared to a baseline system using only PLP features, the sparse representation gives a relative improvement of 5.6% in phoneme error rate on the TIMIT dataset under clean conditions and a 3.2% relative improvement under noisy conditions (additive babble noise).

Sparse coding has also been applied in speaker identification. In [24], a sparse coding representation was used as the input features for several standard classifiers such as Support Vector Machines (SVM) and Gaussian Discriminant Analysis (GDA). The sparse coding representation performed comparably or better than the baseline MFCC and raw spectrogram representations for a variety of noisy test conditions.

The sparse coding model was extended in [82] (where it is referred to as efficient coding) so that basis functions can vary in size, can occur more than once in the coding and can occur in any temporal position. When trained on speech, the learned basis functions resemble the physiological responses obtained from cat auditory nerve fibres. Using the same theoretical model, in [58], a perceptual experiment was performed to compare the intelligibility of speech generated using the sparse coding model against other types of cochlear filters. The study showed that speech generated from the sparse coding model was the most recognisable.

The sparse coding paradigm plays an integral role in helping learning algorithms to achieve biologically plausible representations. Sparsity regularization has also been a crucial ingredient in improving deep learning methods which are reviewed in the rest of this chapter.

2.5 Deep Learning

Artificial neural networks are capable of extracting features based on the underlying statistical properties of the dataset it is trained on. Using a network with multiple hidden layers, it may be possible to obtain a hierarchical feature representation.

Unfortunately, training these deep neural networks with supervised learning and standard error backpropagation often results in worse performance compared to shallow architectures with only one or two hidden layers [5]. This is in large part due to the complexity of deep neural networks with millions of parameters needing to be tuned. Achieving good performance requires lots of labelled data which in certain applications can be limited. Even if the required labelled data is available, the objective function is likely to be non-convex and given the high dimensionality of the parameter

space, there will be lots of poor local minima. Random initialization of parameters will most likely lead to parameters being trapped in one of these poor local minima during training [16].

In 2006, Hinton et al. [33] proposed a new training strategy for the Deep Belief Network (DBN), overcoming the problems discussed above. The training procedure is unsupervised and works by training each layer of the network one at a time as a Restricted Boltzmann Machine (RBM). This procedure is known as pre-training. The weights learnt during pre-training can be used to initialize the weights of a deep neural network which is then trained normally using supervised training, a process known as fine-tuning. Throughout the rest of this report, we will refer to this hybrid learning strategy as the deep learning strategy.

As an alternative to using RBMs as the building blocks for deep architectures, Bengio et al. [5] showed that pre-training can be performed using autoencoders instead of RBMs, resulting in comparable performance [49]. By using this unsupervised pre-training procedure to initialize the weights of a deep architecture, we are starting supervised fine-tuning using a more structured set of weights that has already captured some of the underlying statistics of the input distribution. This puts the weights in a part of the parameter space close to a good local minimum, one that is highly unlikely to be reached through random initialization alone [15].

As the pre-training procedure is unsupervised, it does not require labelled data which can be difficult to obtain. The data used for pre-training does not necessarily have to come from the same distribution that generated the labelled data used in fine-tuning, allowing for different datasets to be used for unsupervised pre-training and supervised fine-tuning. Examples of this include the self-taught framework used in [64] and the use of datasets from multiple languages for unsupervised pre-training in order to boost the performance of an automatic speech recognition system where the target language has limited amounts of transcribed data [83].

In general, deep architectures can be classified by the type of building block used and the connectivity of neurons between layers, either fully connected or locally connected. Fully connected networks are where each neuron is connected to every neuron in the previous layer. Locally connected networks are where neurons are only connected to a small local neighbourhood of neurons in the previous layer.

2.5.1 Restricted Boltzmann Machine Based Architectures

Restricted Boltzmann Machines (RBM) are probabilistic generative models usually consisting of two layers. The simplest form of RBM has one visible layer corresponding to the observed input data and one hidden layer used to explain the observed data. The units in the visible layer can be either stochastic binary units or real-valued units with independent Gaussian noise depending on the type of input data. The units in the hidden layer are stochastic binary units. The visible and hidden layers are fully connected with a symmetric weight matrix and there are no connections between units of the same layer.

The RBM is an energy based model and all possible combinations of visible and hidden unit vectors are assigned an energy through an energy function. The energy function for stochastic binary units in both the visible and hidden layers is as follows:

$$E(\mathbf{v},\mathbf{h}) = -\sum_{i=1}^{|\mathbf{v}|} a_i v_i - \sum_{j=1}^{|\mathbf{h}|} b_j h_j - \sum_{i=1}^{|\mathbf{v}|}\sum_{j=1}^{|\mathbf{h}|} v_i h_j w_{ij} \tag{2.2}$$

where v_i is the binary state of the visible unit i, a_i the corresponding visible unit bias, h_j is the binary state of hidden unit j, b_j is the corresponding hidden unit bias, w_{ij} is the weight between visible unit i and hidden unit j and $|\mathbf{v}|$ and $|\mathbf{h}|$ are the lengths of the visible and hidden unit vectors.

The energy function for real-valued units with independent Gaussian noise is similar:

$$E(\mathbf{v},\mathbf{h}) = \sum_{i=1}^{|\mathbf{v}|} \frac{(a_i - v_i)^2}{2\rho_i^2} - \sum_{j=1}^{|\mathbf{h}|} b_j h_j - \sum_{i=1}^{|\mathbf{v}|}\sum_{j=1}^{|\mathbf{h}|} \frac{v_i}{\rho_i} h_j w_{ij} \tag{2.3}$$

where ρ_i is the variance of the Gaussian noise for visible unit i.

Through the energy function, the joint probability can then be computed as:

$$p(\mathbf{v},\mathbf{h}) = \frac{1}{Z} \mathrm{e}^{-E(\mathbf{v},\mathbf{h})} \tag{2.4}$$

where Z is known as the partition function and is the total sum of energies over all possible pairs of visible and hidden unit vectors in order to normalize the probability distribution:

$$Z = \sum_{\mathbf{v},\mathbf{h}} \mathrm{e}^{-E(\mathbf{v},\mathbf{h})} \tag{2.5}$$

The probability of a visible vector or data example is then computed by summing

over all possible hidden vectors:

$$p(\mathbf{v}) = \frac{1}{Z}\sum_{\mathbf{h}} \mathrm{e}^{-E(\mathbf{v},\mathbf{h})} \tag{2.6}$$

As can be seen from the previous equations, the weights and biases of the RBM defines the probability distribution over the data. In order to train the model, the weights and biases are updated such that the energy of the training data is decreased whilst the energy of all other possible visible vectors is increased.

More specifically, the weights should be updated using the gradient of the log probability of the training data:

$$\frac{\partial \log p(\mathbf{v})}{\partial w_{ij}} = \mathbb{E}\left[v_i h_j\right]_{data} - \mathbb{E}\left[v_i h_j\right]_{model} \tag{2.7}$$

The expectation over the data, $\mathbb{E}\left[v_i h_j\right]_{data}$, can be computed first by setting the visible unit probabilities to that of a training example and then computing the conditional probability of the hidden units:

$$p(h_j = 1|\mathbf{v}) = \sigma(b_j + \sum_{i=1}^{|\mathbf{v}|} v_i w_{ij}) \tag{2.8}$$

where $\sigma(x) = 1/(1+\mathrm{e}^{-x})$ is the sigmoid function.

The expectation over the data, $\mathbb{E}\left[v_i h_j\right]_{data}$, is then computed simply by taking the product of the visible unit value, v_i, and the hidden unit conditional probability, $p(h_j = 1|\mathbf{v})$.

Computing the expectation over the model, $\mathbb{E}\left[v_i h_j\right]_{model}$, is harder and requires the use of a Markov chain Monte Carlo method. As we can easily compute the conditional probability distributions, we can use alternating Gibbs sampling to do this. We initialise the Markov chain once again by setting the visible units to the training example. We can use Eq. 2.8 to then sample the hidden unit state. From this hidden unit state we can then obtain a reconstruction of the data by using a formula in the same form as Eq. 2.8 for binary visible units:

$$p(v_i = 1|\mathbf{h}) = \sigma(a_i + \sum_{j=1}^{|\mathbf{h}|} h_j w_{ij}) \tag{2.9}$$

and for real-valued units:

$$p(v_i = 1|\mathbf{h}) = \mathcal{N}(a_i + \sum_{j=1}^{|\mathbf{h}|} h_j w_{ij}\ , \rho_i) \tag{2.10}$$

where $\mathcal{N}(.\ ,.)$ is a normal distribution with corresponding mean and variance.

From the reconstruction, we can recompute the hidden unit conditional probabilities and we can keep alternating between the visible and hidden unit states until we reach the stationary model distribution. At this point we can then compute $\mathbb{E}\left[v_i h_j\right]_{model}$ in the same way as $\mathbb{E}\left[v_i h_j\right]_{data}$ using the visible and hidden unit probabilities in the final step of the Markov chain. The gradient of the log probability of the training data can then be computed by taking the difference of the two expectations following Eq. 2.7.

In practice however, using alternating Gibbs sampling to obtain $\mathbb{E}\left[v_i h_j\right]_{model}$ takes a very long time to compute. Instead, an approximate algorithm known as Contrastive Divergence [30] is used. In Contrastive Divergence, rather than running the Markov chain until equilibrium is reached, we stop the chain after the first reconstruction and corresponding hidden unit computation step. Using the conditional probabilities from the visible unit reconstruction and the conditional probabilities of the hidden units given the reconstruction, we can again compute the expectation, $\mathbb{E}\left[v_i h_j\right]_{recon}$, through a pair-wise multiplication of the probabilities.

More concretely, the weights of the RBM are updated as follows:

$$\Delta w_{ij} = \varepsilon\left(\mathbb{E}\left[v_i h_j\right]_{data} - \mathbb{E}\left[v_i h_j\right]_{recon}\right) \tag{2.11}$$

where ε is the learning rate. In practice, we would normally use the average gradient computed over a small "mini-batch" of training examples in order to update the weights. For the biases, we only need to consider the corresponding visible or hidden unit's conditional probability as unlike the weights, a bias is only connected to a single unit.

Once an RBM is trained, a deep belief network (DBN) can be constructed by freezing the weights of this layer and training a second RBM hidden layer using the first hidden layer as the visible layer for this new RBM. As illustrated in Fig. 2.6, further layers can be stacked and trained in the same way to build deeper and deeper networks. Once this unsupervised "pre-training" is completed, the deep network can then be trained or "fine-tuned" as a whole using supervised learning on the specific task at

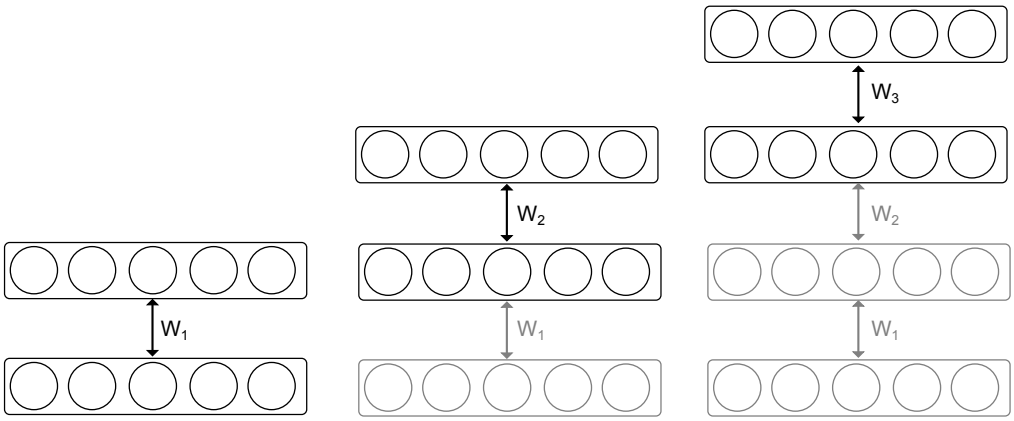


Figure 2.6: Stacking Restricted Boltzmann Machines (RBM) to build deep networks. Each new RBM is trained using the previous hidden layer as input with the weights in the lower layers frozen (shown as greyed out).

hand. In this way, a hierarchical representation of increasing levels of abstraction can be built.

Initially, RBM-based deep networks were used in computer vision tasks, including modelling human motion in video [85] and in a robotic vision system [25]. In [53], DBNs were extended to incorporate sparsity, producing features similar to those found in the human visual system.

Deep networks built from Restricted Boltzmann Machines and their extended variants have also been used successfully in the speech domain, especially on large vocabulary continuous speech recognition (LVCSR) tasks. Methods using deep neural networks have continually set the state-of-the-art in speech recognition as advances in deep neural network modelling continue to grow at a fast pace (see [32] for a comprehensive review).

The hybrid DNN/HMM approach is one commonly used framework applying deep neural networks to LVCSR. In this approach, a deep neural network is used as the acoustic model for a Hidden Markov Model recognizer, replacing Gaussian Mixture Models. In such systems, the deep neural network is trained to estimate the posterior probability of HMM phone states given an acoustic feature observation. Today's deep neural networks are able to directly model context-dependent phone states, with many thousands of states in the output layer.

Key to their success has been the depth of the network with deeper networks learning more invariant and more discriminative features [60] [92] and their ability to exploit information in neighbouring frames [32].

2.5.2 Autoencoder Based Architectures

Autoencoders can be used in place of RBMs in fully connected deep architectures. An autoencoder is a three layer multi-layer perceptron (MLP) consisting of an input layer, a code layer and a reconstruction layer, as illustrated in Fig. 2.7. The goal of an autoencoder is to learn a good code representation that preserves as much information as possible about an input to allow the input to then be reconstructed from the code. An autoencoder is trained unsupervised on a dataset by minimizing a function of the reconstruction error such as the mean squared error.

Autoencoders were originally used for dimensionality reduction where the code layer has fewer units than the input layer, known as a bottleneck layer [75] [4] [13]. It was found that if a linear activation function is used in the code layer, the learned encoding is equivalent to PCA [7]. Typically a non-linear activation function is used such as the sigmoid function, hyperbolic tangent or more recently, the linear rectification function [21]. This allows non-linear features of the dataset to be captured in the encoding.

An encoding, $h(\mathbf{x})$, can simply be computed using the normal feed-forward MLP equation:

$$h(\mathbf{x}) = \mathcal{F}(W\mathbf{x} + \mathbf{b}) \tag{2.12}$$

where $\mathbf{x}$ is the input vector, W is the encoding weight matrix, $\mathbf{b}$ is the bias vector and $\mathcal{F}(.)$ is the activation function. Likewise the reconstruction, $\hat{\mathbf{x}}$ is computed as:

$$\hat{\mathbf{x}} = \mathcal{G}(Dh(\mathbf{x}) + \mathbf{c}) \tag{2.13}$$

where D is the decoding weight matrix, c is the output layer bias vector and $\mathcal{G}(.)$ is the output layer activation function, chosen depending on the scale of the input. The autoencoder can be trained using standard error backpropagation and stochastic gradient descent.

Deep neural networks can be built from autoencoders in much the same way as using RBMs. Autoencoders are stacked on top of one another and trained one layer at a time, with the code layer of the previous autoencoder becoming the input layer of the current autoencoder. Again, all other previously learned weights are held fixed and once unsupervised pre-training has been completed, a further supervised global fine-tuning stage can be performed.

In deep networks, rather than using bottleneck autoencoders as in dimensionality reduction, the code layer typically has many more hidden units than the input. This

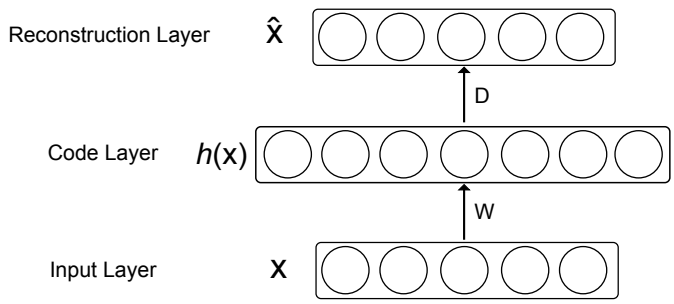


Figure 2.7: Autoencoder with an over-complete code layer.

allows for more complex data distributions to be captured. However, when using these "over-complete" code layers, some form of constraint is required to avoid learning a trivial solution. This could simply be using tied weights (the encoding and decoding weights in the network are the transpose of one another) or imposing a sparsity constraint on the code layer.

Other constraints have resulted in new variants of the autoencoder such as the Denoising Autoencoder (DAE) [88]. The DAE is trained using a stochastically corrupted version of the input and the goal of the DAE is to reconstruct the original noise-free input. This goal of noise removal allows for more robust representations to be learned. Deep architectures built from DAEs were found to perform comparably, if not better than architectures built from RBMs on various image recognition tasks [88] [15].

More recently, Contractive Autoencoders (CAE) have been developed that penalizes the Frobenius-norm of the Jacobian matrix of the encoder. This allows the CAE to learn a locally invariant representation that outperforms DAEs and RBMs on the CIFAR and MNIST image recognition tasks [69].

Instead of using supervised learning for fine-tuning, stacked autoencoders can be "unrolled" or "unfolded" to form a deep autoencoder where each of the encoding transformations are successively undone to reconstruct the input at the final output layer. This process is shown in Fig. 2.8. The deep autoencoder is trained unsupervised as a whole using the reconstruction loss as per a shallow autoencoder.

Fine-tuning as a deep autoencoder ensures that the code representation is still capable of modelling the original input data. During layer-wise pre-training, the focus is on capturing the previous layer's distribution only. This subtle difference in the objective function could decrease the network's ability to model the original input data and fine-tuning the network as a deep autoencoder can help to restore the information loss incurred from the greedy pre-training procedure.

Deep autoencoders can be built from either RBMs or shallow autoencoders and

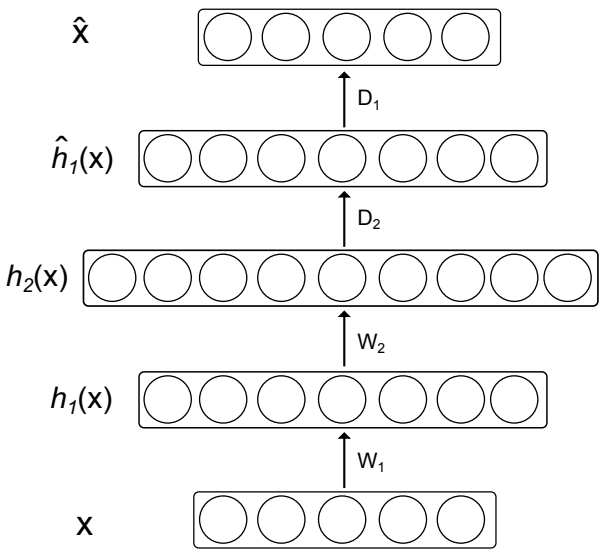


Figure 2.8: Unrolling stacked autoencoders to form a deep autoencoder. Each encoding layer is successively undone using the corresponding decoder weights until the input is reconstructed.

have been used for dimensionality reduction of images [34], learning efficient codes for fast image retrieval [46] and for compressing speech spectrograms [14].

In addition, two identical deep autoencoders coupled at the encoding layer and trained using a special supervised learning objective function have been used to extract a representation containing only the speaker information from MFCCs. This generic speaker-specific representation has been shown to be invariant towards text and language and robust against mismatch conditions in benchmark speaker verification tasks [11] [74].

In the previous section, we discussed the use of the hybrid DNN/HMM framework used in applying deep neural networks to speech recognition. An alternative framework is the TANDEM approach [28] which aimed to combine the discriminative power of neural networks and the many advanced tricks and techniques used in state-of-the-art GMM/HMM speech recognition systems.

In the TANDEM approach, a neural network is trained to discriminate between (context-independent) phone states. However, instead of using the neural network as a replacement acoustic model, the neural network is used as a feature extractor, and these features are used as the input to train a regular GMM/HMM system.

In [28], the log posterior probabilities obtained from a single hidden layer neural network was as used as the input features and provided a performance gain over both the GMM/HMM system with regular acoustic features and the hybrid NN/HMM

system on the Aurora noisy digit recognition task.

An alternative to using posterior probabilities as the input features is to use the output of a bottleneck hidden layer instead. In [23], a neural network with three hidden layers with a middle bottleneck layer was trained as per usual to discriminate between phone states. It was found that using the bottleneck features for the GMM/HMM system outperforms the use of posteriors probabilities on the NIST RT'05 LVCSR meeting transcription task.

The previous two approaches pre-dates the widespread use of deep learning in speech recognition. In [71], deep learning was applied using the TANDEM and bottleneck feature approach. First, a deep neural network with six layers is trained to output the posterior probabilities of phone states. These log posteriors are then used as input to a bottleneck autoencoder with two hidden layers. Using the output of the first deep neural network as input to a second bottleneck autoencoder ensures that the discriminative power of the deep neural network is utilised fully when computing the bottleneck features. Results on the Broadcast News LVCSR task shows that this autoencoder bottleneck approach outperforms both a state-of-the-art GMM/HMM system and hybrid DNN/HMM system.

2.5.3 Convolutional Neural Networks

Prior to Hinton's development of the deep learning strategy, the convolutional neural network (CNN) was a neural network with many hidden layers that could be trained using supervised learning only. This could be done because of the specialized architecture of the CNN. CNNs are inspired by models of the visual system such as Hubel and Wiesel's simple and complex cells [38] and Fukushima's Neocognitron [17]. The CNN is designed especially for visual recognition tasks and consists of many pairs of alternating convolutional and subsampling layers. As an example, the LeNet-5 CNN from [51] is shown in Fig. 2.9.

As shown in Fig. 2.9, the input to a CNN is a 2D image. The first layer of a CNN is a convolutional layer which consists of a set of 2D feature maps. In Fig. 2.9, there are six feature maps in the first convolutional layer, C1. A neuron in a feature map is connected to a small local neighbourhood of the input, known as the receptive field, with adjacent neurons connected to adjacent overlapping neighbourhoods in the input. The number of neurons in a feature map is the number required to cover the whole input image. In C1 in Fig. 2.9, each feature map is of size 28 x 28 with a receptive field size of 5 x 5.

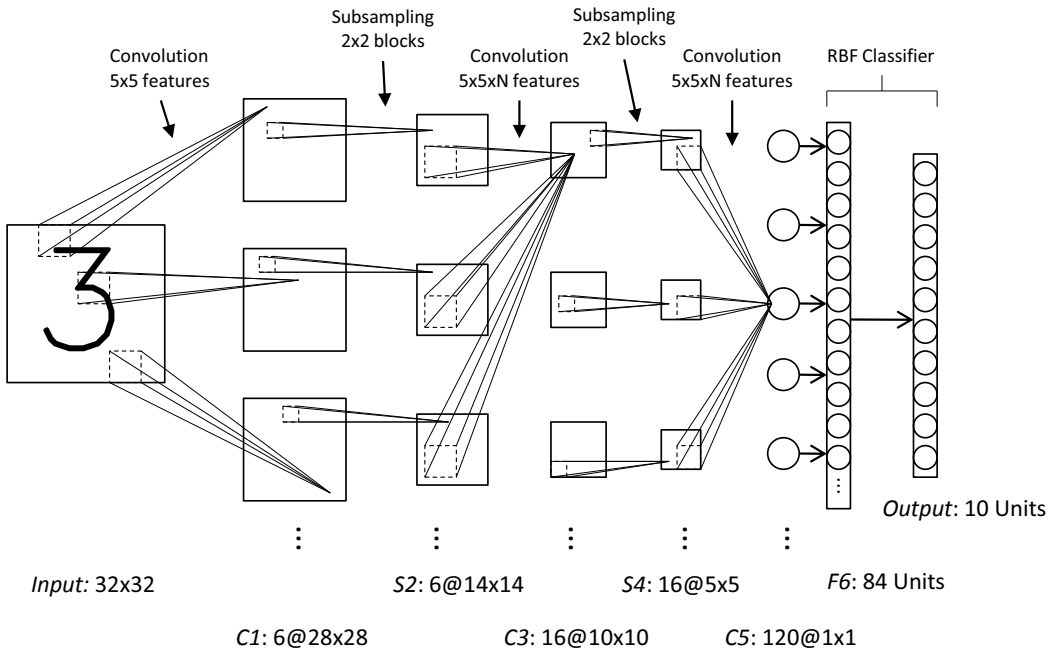


Figure 2.9: LeCun et al.'s LeNet-5 convolutional neural network [51]. The bottom row of labels show the layer number and the size of that layer with X@NxN representing X feature maps of size NxN.

Each of the neurons in a feature map share the same set of weights. By having shared weights, this means that the CNN is searching for a particular feature at all points in the image. The result of this translation invariant feature detection is a 2D map of responses that retains the topological structure of the input image. This weight sharing reduces the number of trainable parameters which helps in successfully training the network using supervised learning alone.

The activation of a neuron is the standard weighted sum of inputs followed by a non-linearity such as the hyperbolic tangent function. From an implementation stand point, all the values in a feature map can be calculated using the convolution operator by convolving the input image with the shared set of weights, also known as the convolution kernel. Following convolution, a non-linearity is applied to each value in the resulting 2D matrix. Multiple convolution kernels are used to in order to detect different features resulting in one feature map per convolution kernel. More specifically, a feature map H_i is computed as:

$$H_i = \mathcal{F}\left(X * W_i + b_i\right) \tag{2.14}$$

where $*$ is the convolution operator, X is a 2D input image, W_i is the convolution kernel for feature map i and b_i is the corresponding bias shared across the whole of feature map i.

Following a convolutional layer is a subsampling layer, where the values of each feature map are subsampled in order to reduce dimensionality and to aid in producing a translation invariant representation. Various subsampling schemes have been used but the most common involves dividing the feature map into small non-overlapping blocks and taking the average or maximum value of each block. These subsampled maps are the input to the subsequent convolutional layer next in the feature hierarchy.

The feature maps in the subsequent convolutional layer are connected to multiple subsampled maps in the previous layer using either a pre-defined connection scheme or a learned connection scheme. Convolution is performed on each of the connected subsampled maps with a separate convolution kernel for each map. The resulting feature maps are summed together and in doing so, the network is trying to compose multiple lower level features into more abstract higher-level features, creating a feature hierarchy of increasingly abstract features. We can generalize Eq. 2.14 to handle multiple input maps:

$$H_i = \mathcal{F}\big(\sum_{j \in S_i} X_j * W_{ij} + b_i\big) \tag{2.15}$$

where S_i is the set of subsampled input feature maps that are connected to H_i.

In Fig. 2.9, a radial basis function (RBF) classifier consisting of the layers F6 and the output layer is stacked on top of the final convolutional layer in the feature hierarchy, C5. This RBF classifier can be switched out for an alternative classifier such as an MLP classifier or a support vector machine.

The features in the CNN are learned by training the network using supervised error backpropagation and stochastic gradient descent, with a slight adjustment to take into account of weight sharing. Although a CNN has many layers, the number of trainable parameters are far fewer than an ordinary fully connected deep neural network. Along with its specialized structure, this allows us to use normal neural network training methods where previously we could not.

CNNs are capable of learning representations that are invariant to pose and lighting [52] and have been used successfully in zip code recognition [50] and as part of an automated cheque reading system [51]. In addition, it has been found that using random convolution kernels in untrained networks seem to perform well on object recognition

tasks showing that the structure of the network plays an important role in its success [40] [76].

More recently, one of the largest scale CNNs ever built achieved state-of-the-art performance on the ImageNet classification task consisting of 1.2 million images and 1000 classes [47]. CNNs have also been adapted for use with the deep learning strategy, details of which follow in the next two sections.

2.5.4 Predictive Sparse Decomposition

Predictive Sparse Decomposition (PSD) [43] was designed to allow convolutional neural networks to take advantage of unsupervised pre-training used as part of the deep learning strategy. PSD uses unsupervised learning to initialize the convolution kernels of a CNN. PSD is an extension of the sparse coding paradigm and tries to solve the problem of inefficient inference.

With sparse coding, once a dictionary has been learned, an expensive optimization procedure is still required to find the sparse representation for a given input. To solve this issue, PSD uses an efficient feed-forward non-linear encoder which is trained jointly with the dictionary to learn a fast approximation of the optimal sparse representation. For inference, the encoder can be used to quickly obtain a good approximation of the optimal sparse representation of the input.

The joint training with a predictive encoder also provides two other benefits, firstly by using a smooth encoding function, the sparse representation obtained is also a smooth function that can be approximated whereas traditional optimization results in highly non-smooth or even non-differentiable functions. Secondly, by joint training with the encoder, a sparse representation can be learned that can capture the specifics of the type of data being modelled, something that a generic optimization method may not normally be able to capture. In particular [44] has learnt features not normally seen with ordinary sparse coding trained on images.

In the original PSD formulation [43], training was performed on image patches randomly sampled from the input dataset. By using image patches the same size as the convolution kernels, the problem can be formulated in terms of a standard fully connected MLP. The predicted encoding of an input image patch is computed using the standard feed forward MLP formula:

$$\mathbf{h}_p = G\tanh(W\mathbf{x} + \mathbf{b}) \tag{2.16}$$

where G is a trainable gain parameter to handle scaling, $\mathbf{x}$ is the vectorized input, W is the weight matrix consisting of the vectorized convolution kernels and $\mathbf{b}$ is the set of biases collected together into a vector.

The predicted encoding is then used in the following three-part loss function for PSD:

$$\mathcal{L}_{\mathcal{PSD}} = \frac{1}{2}||\mathbf{x} - D\mathbf{h}||_2^2 + \lambda||\mathbf{h}||_1 + \alpha||\mathbf{h} - \mathbf{h}_p||_2^2 \tag{2.17}$$

the first part of the loss function deals with the reconstruction loss, where D is the sparse coding dictionary and $D\mathbf{h}$ is the reconstruction of the input from the representation $\mathbf{h}$. The second part is the sparse coding penalty on the representation. The third and final part is the prediction error and constrains the sparse coding representation to be close to the feed-forward predicted encoding.

This loss function is then minimized using a two-step, EM-like procedure:

1. Keeping all the trainable parameters fixed, find the optimal sparse code $\mathbf{h}^*$ using the predicted $\mathbf{h}_p$ as the initial starting value.

2. Using the $\mathbf{h}^*$ found in step 1, update all parameters using one step of stochastic gradient descent.

For step 1, stochastic gradient descent or any other optimization procedure can be used to find the optimal sparse code.

This unsupervised learning procedure can be used for each of the convolutional layers in the CNN. Once this pre-training has been completed, the learned encoder weights can be used to initialize the convolution kernels. The CNN can then be fine-tuned with supervised learning.

Training on image patches independently however, leads to redundancy in the learned features [42]. To counter this, PSD has been extended to incorporate more elements of the convolutional neural network including, altering the sparsity penalty to run over pooled units [42] and training on whole images rather than patches using a convolutional dictionary [44]. These approaches have consistently shown state-of-the-art performance on visual recognition datasets such as MNIST, NORB, Caltech-101 and the INRIA pedestrian detection task.

However, works involving PSD have only used minimal levels of depth with [43], [42], [27] only using a single layer network and [40], [44] using a two layer network. This is most likely due to the difficulty of scaling up convolutional neural networks

when used with realistic sized images. On the other hand, with the rise of GPU computing, such difficulties are fast disappearing as the convolution operator can be efficiently implemented on GPUs as shown by the large scale CNN developed in [47].

2.5.5 Convolutional Restricted Boltzmann Machine

An alternative approach to PSD in combining the deep learning strategy and convolutional neural networks is the Convolutional Restricted Boltzmann Machine (CRBM) [54]. Here we follow the formulation given in [55] where the CRBM was applied to audio data and is the basis of comparison for our work.

The CRBM is very similar to the RBM except for the use of weight sharing. In the convolutional neural network, we have K convolution kernels which we convolve with the input to give K separate feature maps. In the CRBM, the visible layer, which we assume to be a vector of time series data consisting of $|\mathbf{v}|$ frames, is convolved one at a time with the K convolution kernels, $\mathbf{w}_k$, of dimensionality $|\mathbf{w}|$, to give a hidden layer that consists of K groups of feature vectors $\mathbf{h}_k$. The visible units all share a single bias a. Each hidden unit in a feature vector group shares a bias b_k.

To take into account of the weight sharing, the energy function from Eq. 2.2 for binary visible units is now:

$$E(\mathbf{v},\mathbf{h}) = -a\sum_{i=1}^{|\mathbf{v}|} v_i - \sum_{k=1}^{K}\left(b_k \sum_{j=1}^{|\mathbf{h}|} h_{kj}\right) - \sum_{k=1}^{K}\sum_{j=1}^{|\mathbf{h}|}\sum_{r=1}^{|\mathbf{w}|} h_{kj} w_{kr} v_{j+r-1} \tag{2.18}$$

and for real-valued visible units (assuming unit variance):

$$E(\mathbf{v},\mathbf{h}) = \frac{1}{2}\sum_{i=1}^{|\mathbf{v}|} v_i^2 - a\sum_{i=1}^{|\mathbf{v}|} v_i - \sum_{k=1}^{K}\left(b_k \sum_{j=1}^{|\mathbf{h}|} h_{kj}\right) - \sum_{k=1}^{K}\sum_{j=1}^{|\mathbf{h}|}\sum_{r=1}^{|\mathbf{w}|} h_{kj} w_{kr} v_{j+r-1} \tag{2.19}$$

The conditional probability of a hidden unit can be computed using the convolution operator. Eq. 2.8 now becomes:

$$p(h_{kj} = 1|\mathbf{v}) = \sigma(b_k + (R(\mathbf{w}_k) * \mathbf{v})_j) \tag{2.20}$$

where R(.) is a function that rotates a vector by 180°. This is required as the 1D convolution operator will rotate the convolution kernel by 180° first before computing the weighted sum.

The conditional probability of a binary visible unit changes from Eq. 2.9 to:

$$p(v_i = 1|\mathbf{h}) = \sigma(a + \sum_{k=1}^{K} (\mathbf{w}_k * \mathbf{h}_k)_i) \tag{2.21}$$

and for real-valued visible units:

$$p(v_i = 1|\mathbf{h}) = \mathcal{N}(a + \sum_{k=1}^{K} (\mathbf{w}_k * \mathbf{h}_k)_i \, , 1) \tag{2.22}$$

Training the CRBM can be done using the contrastive divergence algorithm as set out in Sec. 2.5.1 with the modified conditional probabilities given above. In addition, the CRBM also requires a sparsity penalty as in [53] due the over-complete representation. This introduces an extra step into the training algorithm where the parameters of the CRBM are updated by the gradient of the sparsity regularization term after the normal parameter updates using contrastive divergence.

Convolutional neural networks normally consist of alternating pairs of convolutional and subsampling layers. As the CRBM is a generative model, a probabilistic version of max-pooling was developed alongside the CRBM [54]. As with deterministic max-pooling, the units in the hidden layer are divided into non-overlapping blocks. For probabilistic max-pooling, within each block, at most one hidden unit can be activated. This is achieved by sampling from a multinomial distribution. Eq. 2.20 changes to:

$$p(h_{kj} = 1|\mathbf{v}) = \frac{\exp(b_k + (R(\mathbf{w}_k) * \mathbf{v})_j)}{1 + \sum_{j' \in B_q} \exp(b_k + (R(\mathbf{w}_k) * \mathbf{v})_{j'})} \tag{2.23}$$

where B_q is the set of hidden units within probabilistic max-pooling block q.

Connected to the hidden layer is an additional pooling layer which again is organised into K groups. A unit in the pooling layer is connected to all of the hidden units within one probabilistic max-pooling block and no others. This shrinks the representation down by a factor of the block size as in deterministic max-pooling in a convolutional neural network. The pooling unit, z_{kq}, can be sampled using the following formula:

$$p(z_{kq} = 0|\mathbf{v}) = \frac{1}{1 + \sum_{j' \in B_q} \exp(b_k + (R(\mathbf{w}_k) * \mathbf{v})_{j'})} \tag{2.24}$$

A convolutional deep belief network (CDBN) can then be built from alternating layers of CRBM and probabilistic max-pooling modules. In [54], a CDBN trained on images of a single object category was shown to learn a hierarchical object-part

representation. When trained unsupervised using four different object categories, the second layer of the learned representation consisted of both object-specific and shared parts whilst the third layer consisted only of object-specific parts. This ability to generate hierarchical representations from unsupervised learning shows the promise of deep convolutional representations in the visual domain.

In [55], the CDBN was applied to audio data. When trained unsupervised on speech data from the TIMIT corpus, the features learned corresponded to phonemes and showed distinct differences between male and female speech. When the features were used for phoneme classification and speaker identification tasks, the CDBN proved competitive against the baseline MFCC representation but falls short when compared against the wider state-of-the-art in both speech recognition and speaker identification.

2.6 Summary

In this chapter, we have explained the basic principles of speech production and speech processing in humans. We have seen how speech can be represented in the form of spectrograms and in the form of low dimensional Mel-Frequency Cepstral Coefficients. We have also seen the potential of deep learning and sparse coding in learning hierarchical feature representations.

Chapter 3

Model Description

In this chapter, we present the details of the convolutional neural network (CNN) building block, the unsupervised learning training methodology and how to build a deep architecture using these CNN building blocks for learning hierarchical speech representations.

3.1 CNN Building Block

A schematic diagram of the CNN building block is shown in Fig. 3.1. The CNN building block is arranged in an encoder-decoder style of architecture with feed-forward connections. Our CNN building block is capable of accepting multiple channels of input vectors but we will concentrate on the simple case of a single input vector and generalize this to multiple input channels in Sec. 3.3.

The CNN takes a vector of time series data, $\mathbf{x}$, of length T, as shown in Fig. 3.1. The convolutional layer consists of M number of feature map vectors. The job of the convolutional layer is to perform feature detection. Each feature vector, $\mathbf{h}_i$, consists of N units. We will denote a unit in $\mathbf{h}_i$ as h_{ij}, where j is the element index of the unit in the i-th feature vector.

Each unit in a feature vector is connected to a small number of K input units. In Fig. 3.1, $K = 3$, meaning each feature vector unit is connected to three input units. In general, a feature vector unit, h_{ij} is connected to the set of inputs $\{x_j, \cdots, x_{j+K-1}\}$. This constrains the size of each feature vector to be $N = T - K + 1$.

Every unit in a given feature vector, $\mathbf{h}_i$, shares the same set of incoming connection weights as illustrated in Fig. 3.1. As each unit is connected to K input units, the shared weight vector, $\mathbf{w}_i$ for the i-the feature vector, $\mathbf{h}_i$, has K elements $[w_{i1}, \cdots, w_{iK}]$.

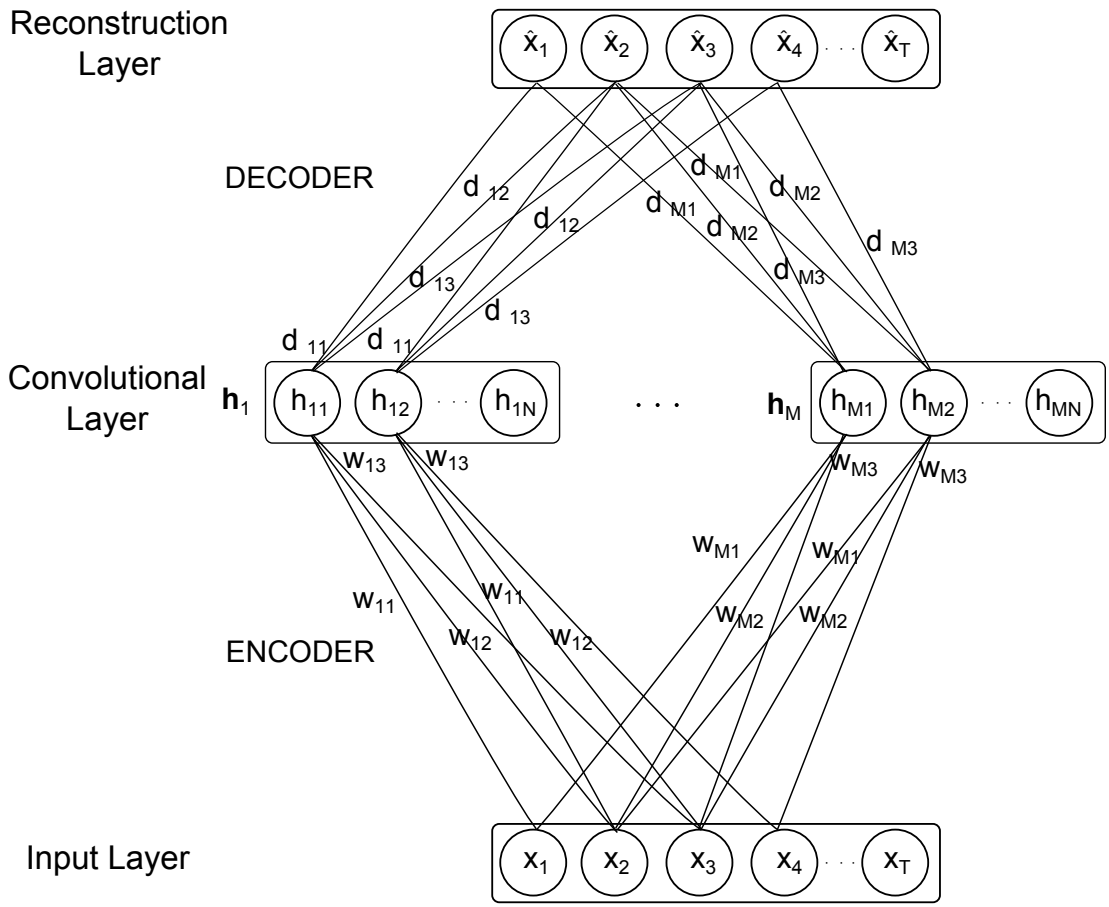


Figure 3.1: Schematic diagram of the CNN Building Block.

The local connectivity means that the network is trying to analyze the local structure of the input and detect small scale local features encoded by the weight vectors, $\mathbf{w}$. The shared weights across each feature vector means that feature detection is temporally invariant as all points in the input are analyzed for the presence of the local feature. Each weight vector is different for each feature vector in order to detect different features, i.e. $\mathbf{w}_i \neq \mathbf{w}_j, \forall i, j$.

The values of each feature vector unit, h_{ij}, can be computed using the weighted sum of input connections in the normal feed-forward MLP fashion and then applying a non-linearity. More conveniently, we can calculate all the values for a feature vector $\mathbf{h}_i$ using the convolution operator:

$$\mathbf{h}_i = g_i \tanh\left([\mathbf{x} *_v R(\mathbf{w}_i)] + \mathbf{b}_i\right) \tag{3.1}$$

where g_i is a trainable gain value used to take into account of the scaling performed by tanh, $\mathbf{b}_i$ is a trainable bias vector, $R(.)$ is a function that rotates a vector by 180° and $*_v$ is the convolution operator in "valid" border handling mode, i.e. we only want the values where the weight vector $\mathbf{w}_i$ fits inside the boundaries of the input vector $\mathbf{x}$. As we are using convolution in valid border mode, we must rotate the weight vector by 180° in order to ensure the weights are applied to the correct connections.

It should also be noted that as well as sharing weights, each unit in a feature vector $\mathbf{h}_i$ also shares the same bias. This means that the bias vector $\mathbf{b}_i$ actually contains one bias value replicated N times for each of the N h_{ij} units.

Having obtained the feature vector representations in the convolutional layer, we attempt to reconstruct the input from the feature vectors in the reconstruction layer. Each h_{ij} unit has symmetric outgoing connections to the reconstructed input, $\hat{\mathbf{x}}$ as shown in Fig. 3.1. Again, each h_{ij} unit in feature vector $\mathbf{h}_i$ shares a set of outgoing decoder weights $\mathbf{d}_i$ with K elements. We can compute the reconstruction using convolution and summing the resulting values as shown below:

$$\hat{\mathbf{x}} = \sum_{i=1}^{M} \mathbf{h}_i *_{full} \mathbf{d}_i \tag{3.2}$$

where $*_{full}$ is the convolution operator in "full" border handling mode. Note we do not need to rotate the decoder weights as we did in Eq. 3.1. Eq. 3.2 approximates the input $\mathbf{x}$ as a linear combination of the decoder weights $\mathbf{d}$.

To learn the weights and other parameters of our CNN building block, we used the unsupervised Predictive Sparse Decomposition (PSD) algorithm [43] (reviewed in

Sec. 2.5.4) and described in the next section.

3.2 Predictive Sparse Decomposition Learning

Predictive Sparse Decomposition (PSD) allows us to learn the parameters of our CNN building block in an unsupervised way. In particular, we are interested in learning the weights for the encoder part of the CNN building block as it computes the feature representation. We use PSD because traditional sparse coding [62] only allows us to learn the decoder weights, whereas PSD allows us to learn both.

Rather than training on the input as a whole, we follow the original PSD formulation and train on patches of the input the same size as the weight vectors $\mathbf{w}$. We do not use the extensions of PSD as they are even more specific to learning in the visual domain.

Due to the local connectivity of the CNN, we then only need to compute, for each of the M feature vectors, the value of the corresponding single h_{ij} unit that is connected to the input patch $\mathbf{x}_j$, where j is the index of the patch's first element in $\mathbf{x}$. To compute the value of each h_{ij} unit, rather than using convolution, we can now use simple matrix arithmetic like we would in a normal MLP setting. We modify Eq 3.1 to become:

$$\mathbf{h}_j = G\tanh(\mathbf{x}_j W + \mathbf{b}) \tag{3.3}$$

where $\mathbf{h}_j$ is the resulting vector containing each h_{ij} value $\forall i \in M$; G is a diagonal gain matrix containing each g_i value along the diagonal; W is an M x K matrix with each weight vector $\mathbf{w}_i$ on the i-th row of W and $\mathbf{b}$ is a vector of biases containing each individual shared bias $b_i, \forall i \in M$.

Given the hidden vector $\mathbf{h}_j$, we can reconstruct the input patch in a manner similar to Eq. 3.2.

$$\hat{\mathbf{x}}_j = D\mathbf{h_j} \tag{3.4}$$

where D is a K x M matrix containing each decoder weight vector $\mathbf{d}_i$ along the i-th column of D. In [43], each $\mathbf{d}_i$ is referred to as a basis function.

To learn the parameters, we need to minimize the following loss function:

$$\mathcal{L} = \frac{1}{2}||\mathbf{x}_j - D\mathbf{h}_j^*||_2^2 + \lambda||\mathbf{h}_j^*||_1 + \frac{\alpha}{2}||\mathbf{h}_j^* - \mathbf{h}_j||_2^2 \tag{3.5}$$

where $\mathbf{h}_j^*$ is the optimal sparse hidden representation, $\mathbf{h}_j$ is the hidden vector obtained

from our encoder in Eq. 3.3, λ is the sparsity parameter and α is a parameter controlling the contribution of the third term. The first term of Eq. 3.5, penalizes inaccurate reconstructions, the second term penalizes non-sparse representations controlled by λ, whilst the third term drives the encoder towards producing the optimal representation controlled by α.

For each input patch $\mathbf{x}_j$ in the training set, we perform training in two stages. In the first stage, we try to find $\mathbf{h}_j^*$ for the current set of parameters. We do so by using the $\mathbf{h}_j$ obtained from the encoder, Eq. 3.3, as a starting point for minimizing the loss function, Eq. 3.5, using stochastic gradient descent. The update rule is as follows:

$$\mathbf{h}_j^* \leftarrow \mathbf{h}_j^* - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{h}_j^*}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}_j^*} = -D^T(\mathbf{x}_j - D\mathbf{h}_j^*) + \lambda sign(\mathbf{h}_j^*) + \alpha(\mathbf{h}_j^* - \mathbf{h}_j) \tag{3.6}$$

where η is the learning rate.

In the second stage, using the value of $\mathbf{h}_j^*$ obtained from the first stage, we update the parameters using one step of stochastic gradient descent as follows. Firstly, the weights:

$$W \leftarrow W - \varepsilon \frac{\partial \mathcal{L}}{\partial W} \tag{3.7}$$

$$\frac{\partial \mathcal{L}}{\partial W} = -\alpha\big(g'(\mathbf{x}_j)(\mathbf{h}_j^* - \mathbf{h}_j)\big)\mathbf{x}_j \tag{3.8}$$

where $g'(\mathbf{x}_j) = \frac{1}{G}(G^2 - \mathbf{h}_j^2)$ and ε is the encoder learning rate.

Next the encoder biases:

$$\mathbf{b} \leftarrow \mathbf{b} - \varepsilon \frac{\partial \mathcal{L}}{\partial \mathbf{b}} \tag{3.9}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}} = -\alpha\big(g'(\mathbf{x}_j)(\mathbf{h}_j^* - \mathbf{h}_j)\big) \tag{3.10}$$

The encoder gain parameters:

$$G \leftarrow G - \varepsilon \frac{\partial \mathcal{L}}{\partial G} \tag{3.11}$$

$$\frac{\partial \mathcal{L}}{\partial G} = -\alpha(\mathbf{h}_j^* - \mathbf{h}_j)(\tanh(W\mathbf{x}_j + \mathbf{b})) \tag{3.12}$$

Finally, the decoder weights:

$$D \leftarrow D - \mu \frac{\partial \mathcal{L}}{\partial D} \tag{3.13}$$

$$\frac{\partial \mathcal{L}}{\partial D} = -(\mathbf{x}_j - D\mathbf{h}_j^*)(\mathbf{h}_j^*)^T \tag{3.14}$$

where μ is the decoder learning rate. Each column of D must then be scaled to unit norm to avoid learning trivial solutions.

This two stage process is repeated for all input patches in the training set, by the end of which, the learned encoder parameters should produce a close approximation to the optimal sparse hidden representation.

3.3 Learning Hierarchical Representations

We can stack multiple CNN building blocks together and using the deep learning strategy [33], we can create a hierarchical feature representation. Before doing so however, we perform one further processing step on all feature vectors $\mathbf{h}$ in the convolutional layer after the encoder parameters have been learned through PSD. For a given input $\mathbf{x}$, we compute all feature vectors $\mathbf{h}$ using Eq. 3.1. We then subsample the values in each feature vector in order to reduce dimensionality and to provide further temporal invariance.

We subsample using a technique called max-pooling [68] [65]. For each feature vector $\mathbf{h}_i$, we divide the N units into distinct non-overlapping groups of size S. Within each group, we take the maximum value, resulting in a max-pooled vector of length $P = N/S$. Where P is not an integer number, we zero-pad the beginning and end of the input $\mathbf{x}$ so that P results in a whole number.

We use the max-pooled feature vectors, which we denote collectively as $\mathbf{p}$, as the feature representation for the input $\mathbf{x}$. We can create a feature hierarchy by stacking CNNs and training each on the previous layer's feature representation $\mathbf{p}^{l-1}$, where l is the layer being trained. As there are now M^{l-1} vectors as input to the convolutional layer l, we need to generalize Eq. 3.1 so that it can accept multiple input vectors:

$$\mathbf{h}_i^l = g_i^l \tanh\left(\sum_{q=1}^{M^{l-1}} [\mathbf{p}_q^{l-1} *_v R(\mathbf{w}_{iq}^l)] + \mathbf{b}_i^l\right) \tag{3.15}$$

where $\mathbf{p}^0 = \mathbf{x}$. There now exists a weight vector $\mathbf{w}_{ij}$ for each of the input vectors $\mathbf{p}_q$

which are still of size K and are shared across each feature vector unit in a feature vector $\mathbf{h}_i$. The summation in Eq. 3.15 means that in the higher levels of the feature hierarchy, we are trying to combine lower-level features into more abstract composite features, giving us a hierarchical representation of increasing levels of abstraction.

We also need to make some adjustments to the PSD algorithm to take into account of multiple input vectors. We construct the input vector $\mathbf{x}_j$ by extracting patches of size K^l from each of the $\mathbf{p}^{l-1}$ vectors and concatenating them into a single column vector. This results in the length of $\mathbf{x}_j$ being equal to K^l x M^{l-1}. The patches must also be extracted from the same corresponding region of each $\mathbf{p}_q^{l-1}$, i.e. the starting element of the patch must be j-th element in each $\mathbf{p}_q^{l-1}$.

We must also concatenate each $\mathbf{w}_{iq}^l$ weight vector into a corresponding row vector $\mathbf{w}_i^l$. We can then form the weight matrix W as per usual. Likewise, we must do the same for each $\mathbf{d}_{iq}^l$ decoder weight vector to form the matrix D. Learning can then proceed as normal as described in the previous section to learn the parameters for layer l in the hierarchical representation.

Chapter 4

Comparison Study

In this chapter, we provide the details of the model settings that we used for learning a hierarchical representation from speech data using the CNN building block described in the previous chapter. We compare our model against Lee et al.'s CRBM results and MFCC baseline presented in [55]. The experiments performed were gender classification, speaker identification and phoneme classification using the TIMIT corpus [18]. In our experiments, we take care to try and match as closely as possible the experimental settings in [55] to make for a fair comparison. We also perform a qualitative comparison of our model's learned features against the CRBM features presented in [55].

4.1 Model Settings and Training

Following [55], we used the TIMIT [18] training set to draw unlabelled data to train our CNN architecture with. The TIMIT training set consists of 4620 utterances from 462 speakers, each contributing 10 utterances. Each utterance is on average 3 seconds in duration.

For each utterance, we used the same pre-processing technique as [55]. Each utterance was converted into a spectrogram using a 20ms Hamming window with 10ms overlap. The spectrogram was PCA whitened to reduce the dimensionality of the spectrogram, retaining the first 80 principal components. This resulted in 80 time-series vectors as input to our CNN with P_0 frames in length.

To make our results as comparable as possible, we used the same sized architecture as [55] and extracted a two layer feature representation using our CNN architecture. The first convolutional layer consists of $M = 300$ feature vectors, giving 300 x 80

weight vectors, each with size $K = 6$ frames in length. Following convolution, each feature vector, $\mathbf{h}_i$, was max-pooled using a block size of $S = 3$ to give 300 max-pooled vectors, $\mathbf{p}_i^1$, each with $P_1 = (P_0 - K + 1)/S$ frames in length. These max-pooled vectors forms the full first layer representation. We denote this as the full L1 representation.

In the second layer, following [55], we also used $M = 300$ and $K = 6$ resulting in 300 x 300 weight vectors. Again, following convolution, we max-pooled with a block size of $S = 3$, to produce 300 $\mathbf{p}_i^2$ vectors of length $P_2 = (P_1 - K + 1)/S$, forming the full L2 representation.

We used the procedure described in the previous chapter (Sec. 3.2 and 3.3) to train our architecture. Details of the implementation of the training algorithm can be found in Appendix A. We first trained the L1 representation using a training set constructed by randomly extracting 50,000 patches of size 80 x 6 from the PCA whitened spectrograms of the utterances in the TIMIT training set. Each patch was vectorized as per the training procedure. We further extracted 10,000 patches randomly to form a validation set. Hyperparameters were selected based on the performance of the model on the validation set, evaluated using the loss function in Eq. 3.5.

Once the L1 representation was learnt, we trained the second layer CNN on the L1 representations of each utterance. From the full L1 representations, we constructed a training set by randomly extracting 25,000 patches and a validation set by randomly extracting a further 10,000 patches. Again, hyperparameter selection was based on the model's performance on the validation set evaluated using Eq. 3.5.

We then tested our learned representations on three speech classification tasks. Following [55], we used a linear SVM classifier for each classification task. Where a task is a multi-class classification problem, we used a multi-class SVM in one vs one mode. More specifically, we used the LibSVM Matlab implementation [10].

Since each utterance is of variable length, the size of our feature representations are not uniform across the dataset, which poses a problem as the SVM classifier requires fixed sized feature vectors. In the interests of a fair comparison, we used the approach in [55] and used simple summary statistics to reduce our variable length representations to a fixed size. For example, our full L1 representation consists of 300 $\mathbf{p}_i^1$ vectors, each of length P_1. For each $\mathbf{p}_i^1$, we take for instance, the average over all P_1 values, resulting in a 300 x 1 vector as input to the SVM.

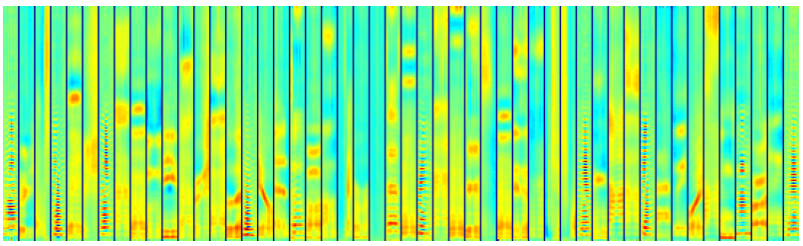
The summary statistics we trialled were the mean, maximum and standard deviation. The choice of summary statistic and hyperparameters for the SVM classifier were chosen using either cross-validation or a validation set. Details of this and the

results of our experiments are presented the following sections but firstly we perform a qualitative analysis of our learned representation.

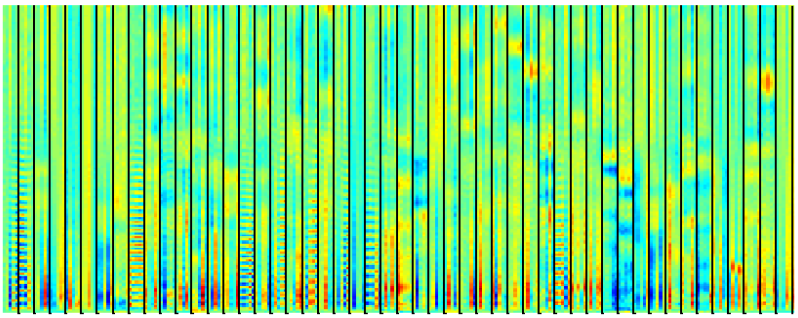
4.2 Visualization

In this section, we compare visualizations of our model's learned weights against the CRBM visualizations presented in [55]. Firstly, we present a random selection of 50 first layer weights as shown in the bottom half of Fig. 4.1. The top half of Fig. 4.1 shows 50 random first layer weights for the CRBM taken directly from [55]. These visualizations were simply produced by reversing the PCA preprocessing on each weight vector.

We can see that our model's learned weights consist of a mixture of harmonic detectors (horizontal bands) useful for detecting voiced sounds, whilst others are looking for concentrations of energy particularly in the low frequency ranges. Compared with the CRBM features, there are also harmonic detectors but their features more clearly show the detection of formant trajectories compared to our CNN features.



(a) CRBM



(b) CNN

Figure 4.1: A random selection of 50 first layer weights. (a) CRBM visualization taken from [55]. (b) Visualization of our CNN model's weights.

Next, we compare the first layer weights for four phonemes, "ah", "oy", "el" and "s". For each phoneme, we show the weight with the largest corresponding activation.

In each figure (Fig. 4.2 - 4.5), part (a) shows the spectrogram of the phoneme from [55] whilst part (b) shows the corresponding first layer CRBM weight with the largest activation beneath it. Parts (c) and (d) show the equivalent for our model's first layer weights.

From Fig. 4.2 - 4.5, we see that in general our model has learned to capture some sensible aspect of the input. For the "ah" phoneme in Fig. 4.2, we see a mixture of harmonic detectors and detection of formant energies similar to the CRBM features. For the "oy" phoneme in Fig. 4.3, we see a similar pattern in our learned CNN features, although the CRBM features are more capable of tracking the rising formant.

In the "el" phoneme example in Fig. 4.4 we see a noticeable difference between our features and CRBM features. The CRBM features prefer to pick out the general spectral shape whereas our features either focuses on the detection of harmonics (2nd, 4th and 5th examples) or formants (1st and 3rd examples).

Finally, for the unvoiced "s" phoneme in Fig. 4.5, we see that the for our CNN features, the first and the second examples result in the same weight being maximally active with the third example activating a very similar looking weight. This shows some robustness in the representation that same weights are maximally active for different examples of the same phoneme.

The final visualization comparison involves comparing how male and female versions of the "ae" phoneme are represented in each of the models. The first comparison shows the first layer weights in Fig. 4.6. The second comparison shows the second layer weights in Fig. 4.7. For visualizing the second layer weights, we unfold the deep architecture to form a deep autoencoder and show the model's reconstruction of the input using only the second layer weight. We note that unlike for the CRBM, our encoding and decoding weights are not the same but nevertheless, the reconstruction serves as a useful indication of what the model has learned in the higher layers.

In part (a) of both Fig. 4.6 and 4.7, five example spectrograms of female speakers speaking the "ae" phoneme are shown (these are same across both figures for ease of comparison) taken from [55]. Likewise, part (b) of both figures show the male counterparts. Parts (c) and (d) show what are described as the most "biased" CRBM features toward each gender taken from [55]. Parts (d) and (e) show our model's top five features that activate when averaged over all "ae" phoneme examples for each gender.

As with the CRBM features, in Fig. 4.6, we can see a distinct difference between the first layer weights in our model that are most active for female speakers and male

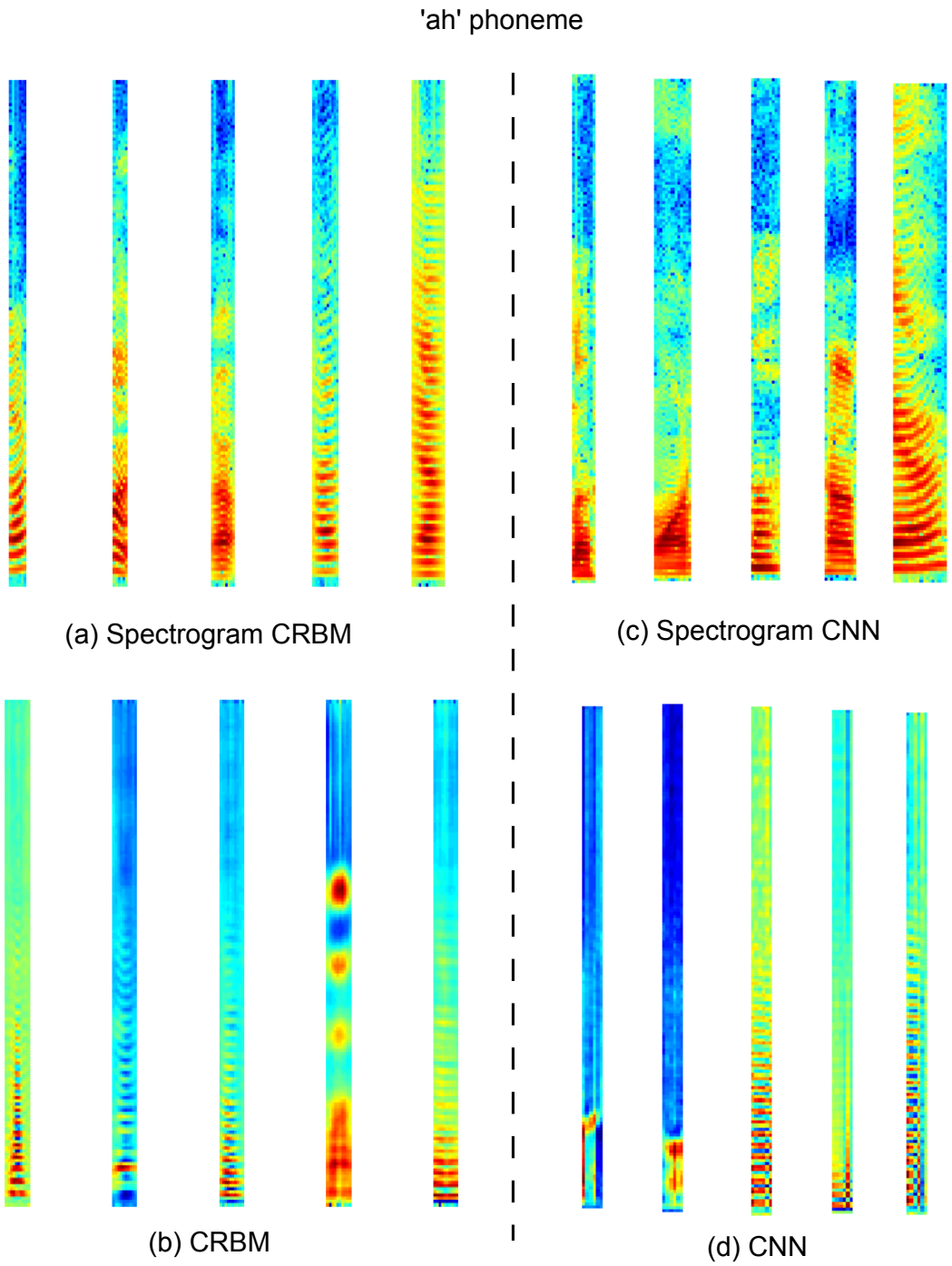


Figure 4.2: Comparison of the most active first layer weights for five examples of the "ah" phoneme. (a) Spectrogram of each of the five example "ah" phonemes taken from [55]. (b) The most active first layer CRBM weight when the above example phoneme is used as input taken from [55]. (c) and (d) Same as for (a) and (b) but using our model's first layer CNN weights.

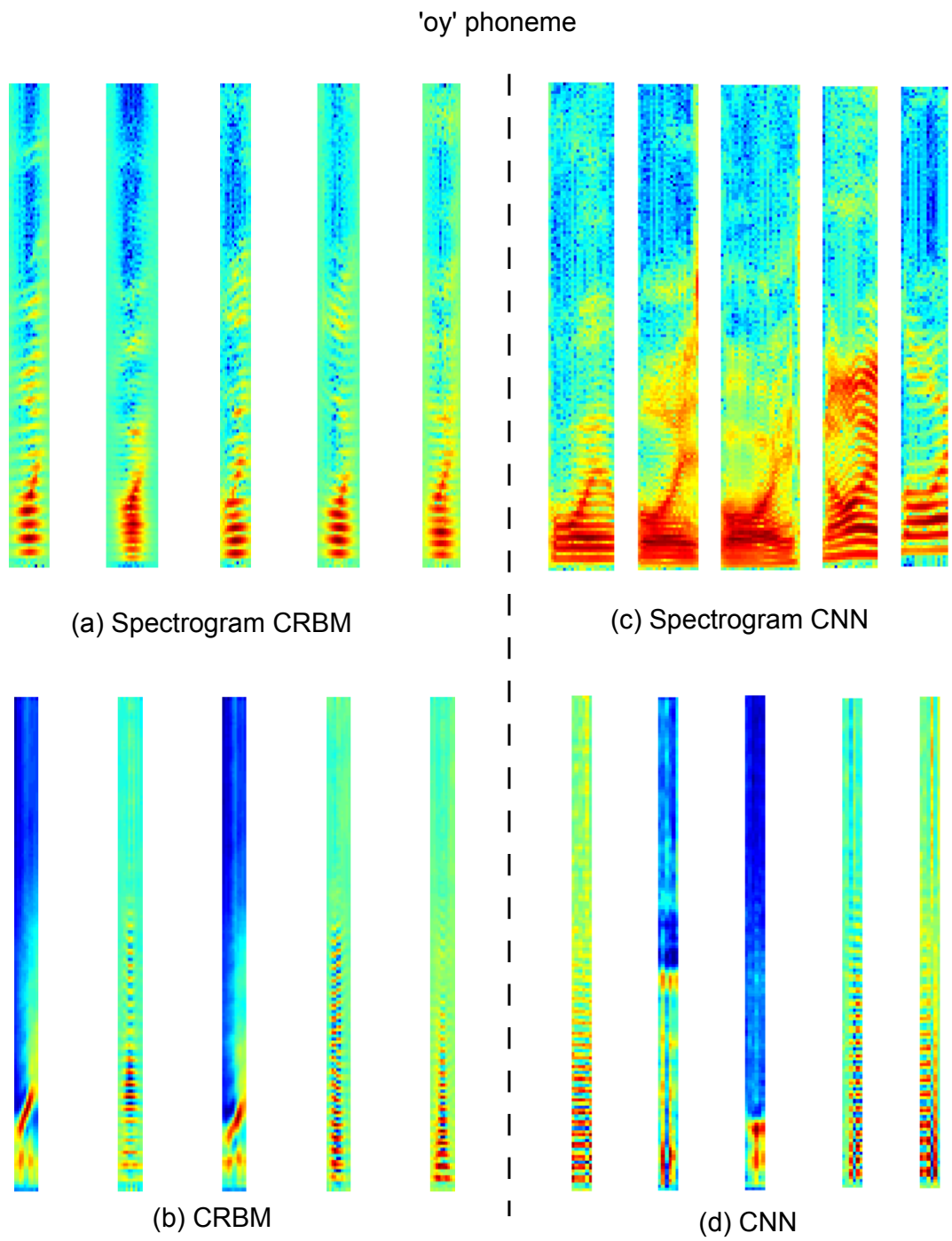


Figure 4.3: Comparison of the most active first layer weights for five examples of the "oy" phoneme. (a) Spectrogram of each of the five example "oy" phonemes taken from [55]. (b) The most active first layer CRBM weight when the above example phoneme is used as input taken from [55]. (c) and (d) Same as for (a) and (b) but using our model's first layer CNN weights.

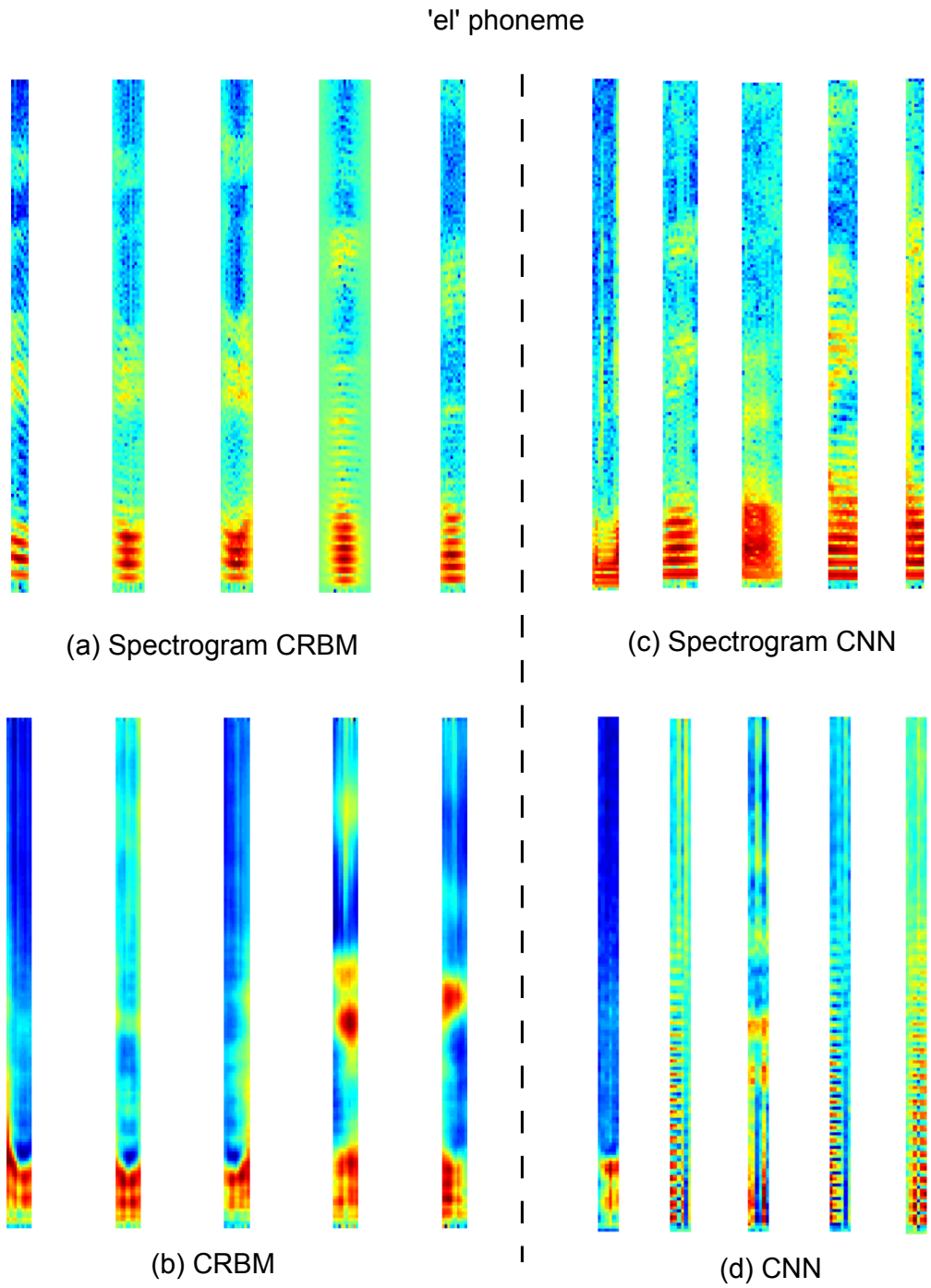


Figure 4.4: Comparison of the most active first layer weights for five examples of the "el" phoneme. (a) Spectrogram of each of the five example "el" phonemes taken from [55]. (b) The most active first layer CRBM weight when the above example phoneme is used as input taken from [55]. (c) and (d) Same as for (a) and (b) but using our model's first layer CNN weights.

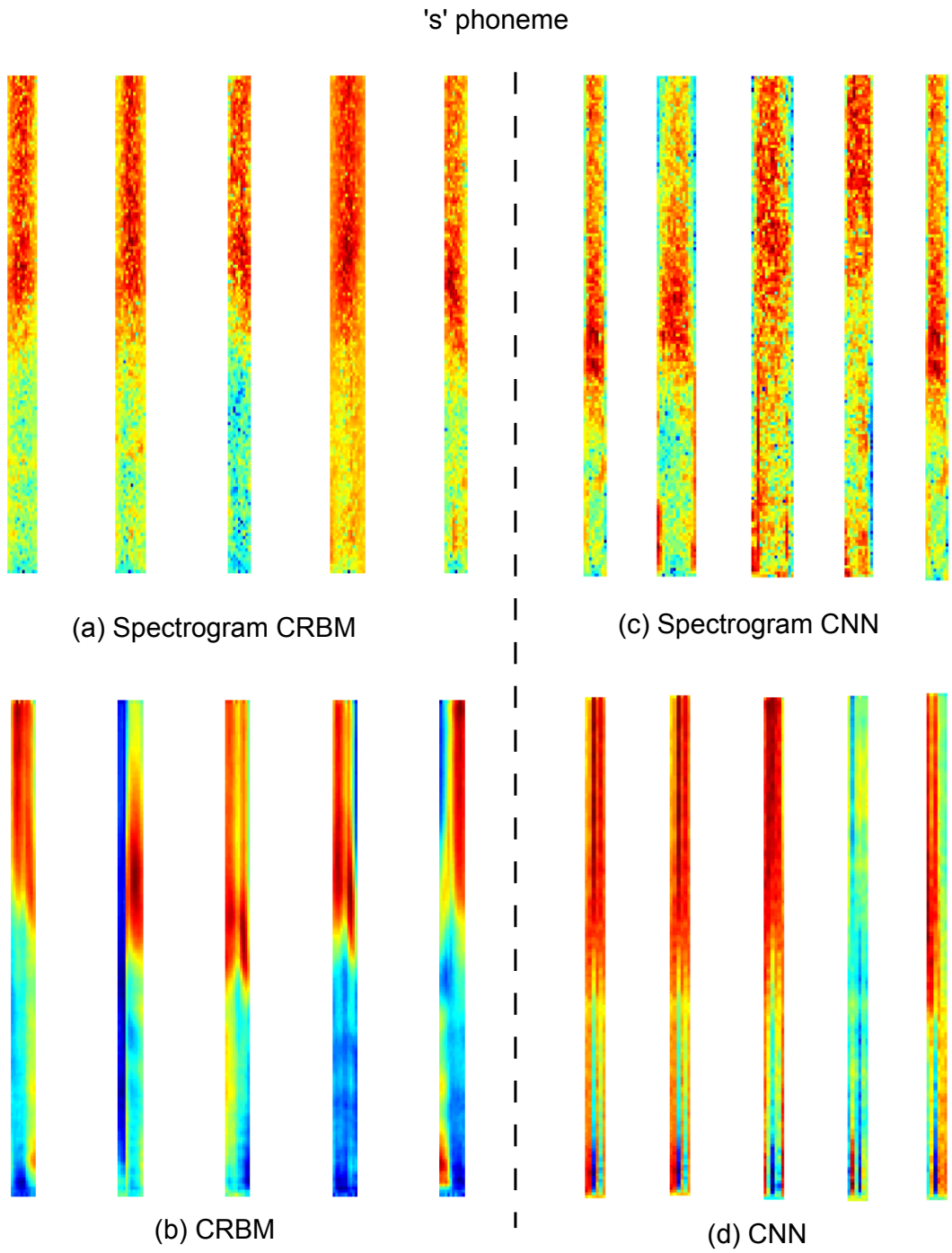


Figure 4.5: Comparison of the most active first layer weights for five examples of the "s" phoneme. (a) Spectrogram of each of the five example "s" phonemes taken from [55]. (b) The most active first layer CRBM weight when the above example phoneme is used as input taken from [55]. (c) and (d) Same as for (a) and (b) but using our model's first layer CNN weights.

speakers. For female speakers, the model tends to concentrate on the harmonic pattern whilst for male speakers, the formants are most preferred. As we've seen previously, the CRBM features tend show more of a trajectory than our CNN features.

For the second layer weights in Fig. 4.7, there is no noticeable gender difference in our CNN features unlike the CRBM features. We hypothesise that our second layer features have learned to become more invariant towards speaker attributes such as gender. This hypothesis is tested in the gender classification experiment in the following section.

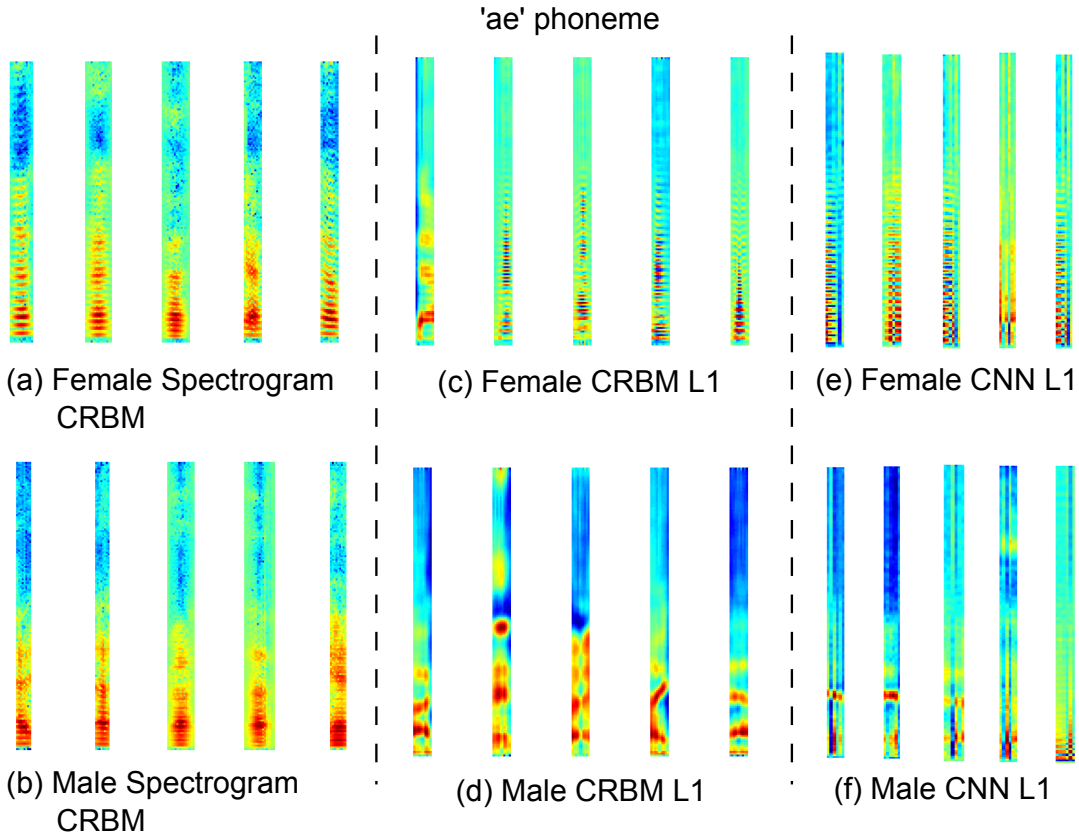


Figure 4.6: Comparison of gender encoding in each model with the "ae" phoneme. (a) Five example spectrograms of female speakers taken from [55] (b) Five example spectrograms of male speakers taken from [55] (c) The five most biased first layer CRBM weights for female speakers taken from [55] (d) The five most biased first layer CRBM weights for male speakers taken from [55] (d) The five most active first layer CNN weights for female speakers (e) The five most active first layer CNN weights for male speakers

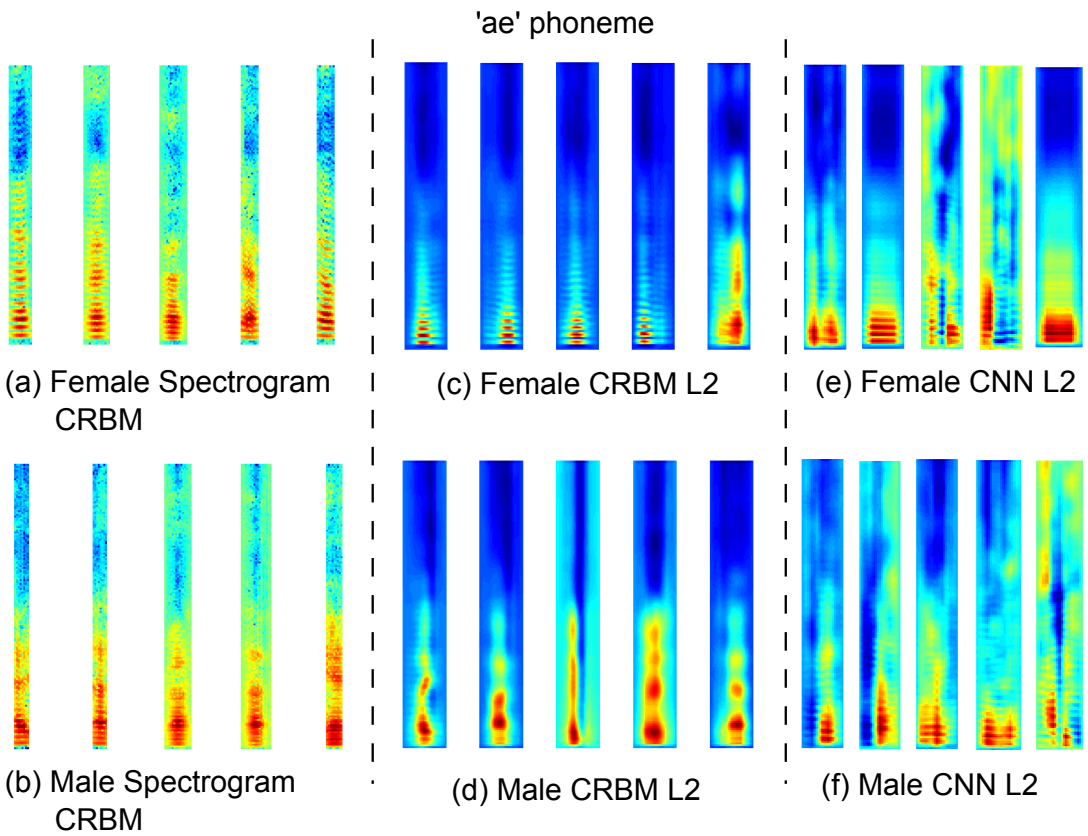


Figure 4.7: Comparison of gender encoding in each model with the “ae” phoneme. (a) and (b) The same ten example spectrograms from Fig. 4.7 taken from [55] (c) The five most biased second layer CRBM weights for female speakers taken from [55] (d) The five most biased second layer CRBM weights for male speakers taken from [55] (d) The five most active second layer CNN weights for female speakers (e) The five most active second layer CNN weights for male speakers

4.3 Gender Classification

In the gender classification experiment, the task is to classify whether a given utterance is spoken by a male or female speaker. For this task, we tested both L1 and L2 feature representations. We trained the SVM classifier using a variable number of training utterances per gender. The training set was constructed by randomly sampling utterances from the TIMIT complete test set. The TIMIT complete test set contains 1680 utterances from 168 speakers with a 2:1 male-female ratio.

We tested using a randomly constructed test set of 200 utterances as per [55]. We report the average classification accuracy over 20 train-test runs. Hyperparameter selection and choice of summary statistic was performed using a validation set of 25 utterances per gender, randomly sampled from the separate TIMIT training set which was previously used for unsupervised training only. Table 4.1 shows a comparison of our results with [55] and the MFCC baseline.

Table 4.1: Average test accuracy on gender classification

# training utterances	CNN L1	CNN L2	CRBM L1 [55]	CRBM L2 [55]	MFCC [55]
1	**90.2**%	82.7%	78.5%	85.8%	58.5%
2	**94.1**%	86.1%	86.0%	92.5%	78.7%
3	**94.9**%	88.1%	88.9%	94.2%	84.1%
5	94.3%	89.3%	93.1%	**95.8**%	86.9%
7	96.1%	92.5%	94.2%	**96.6**%	89.0%
10	96.3%	91.9%	94.7%	**96.7**%	89.8%

Our results were obtained using the maximum summary statistic. As shown in Table 4.1, our learned representation outperforms the baseline MFCC representation. We outperform [55] for smaller numbers of training utterances and is comparable for larger numbers.

The second point to notice is that our L1 representation outperforms our L2 representation, whilst the opposite is true in [55]. This is consistent with the visualization of the most active weights for each gender in Fig. 4.6 and 4.7.

4.4 Speaker Identification

In speaker identification, the task is to identify the speaker of a given utterance from a given set of speakers. This is a *closed-set* task meaning that the model will not be tested

against any unknown speakers. We tested both our L1 and L2 representations using the TIMIT complete test set. As mentioned in the previous section, the complete test contains 168 speakers with 10 utterances per speaker. We treat this task as a 168-way classification problem.

Following [55], we randomly selected varying numbers of training utterances per speaker and tested (separately) on two random utterances per speaker, except in the case of eight training and two testing utterances per speaker. In this case, we used the two "sa" sentences, three "si" sentences and the first three "sx" sentences for training and the remaining two "sx" sentences for testing. We performed 10 train-test runs and report the average classification accuracy in Table 4.2. Hyperparameter and summary statistic selection were chosen using cross-validation, except in the case of one training utterance per speaker where a randomly chosen validation sentence per speaker was used.

Table 4.2: Average test accuracy on speaker identification

# training utterances	CNN L1	CNN L2	CRBM L1 [55]	CRBM L2 [55]	MFCC [55]
1	65.5%	58.5%	**74.5**%	62.8%	54.4%
2	**78.4**%	69.3%	76.7%	66.2%	69.9%
3	88.0%	80.2%	**91.3**%	84.3%	76.5%
5	**93.8**%	88.0%	93.7%	89.6%	82.6%
8	97.3%	92.3%	**97.9**%	95.2%	92.0%

Our results were obtained using the mean summary statistic. Overall, as shown in Table 4.2, our representation is generally worse than [55]. We believe this is because of the patch based nature of the PSD algorithm. With our architecture, PSD trains on patches six frames in duration which is equivalent to approximately 70ms of speech. With such short durations of speech, it would be difficult for any algorithm to learn to model speaker dynamics well. In [55], the whole utterance (approximately 3 seconds in duration) is used for feature learning. It is more likely that the PSD algorithm learns to model phonemes as each patch corresponds to approximately to the average phoneme duration. This hypothesis is tested in the next task.

Despite this, our representation still performs better than the MFCC baseline. This provides some evidence for using deep learning features over MFCC based features for speaker related tasks.

4.5 Phoneme Classification

In phoneme classification, the task is to classify a given phoneme segment from an utterance as one of 39 phoneme classes. (This is distinct from phoneme *recognition* where an utterance must be segmented and labelled.) We followed the phoneme clustering in [55] to give 39 phoneme classes. As with [55] we tested our L1 representation and used the "si" and "sx" sentences in the TIMIT training set for training our SVM classifier and the TIMIT core test set is used for testing.

We treated each phoneme segment as an individual example and computed the L1 representation for each. We did so by computing the PCA whitened spectrogram for an utterance and segmenting the phonemes using the annotations provided by the corpus. Where the number of frames did not naturally fit our CNN architecture, i.e. the length of $\mathbf{h}_i$ is not divisible by the max-pooling block size of three, we extracted a sufficient number of frames centred on the phoneme segment. This means that at most 40ms of extra speech is extracted before and after the phoneme segment.

We report the average classification accuracy over 5 runs for varying numbers of training utterances, except in the case of 3696 utterances where all the training data is used. The results are shown in Table 4.3. We used cross-validation for hyperparameter and summary statistic selection.

Table 4.3: Average test accuracy on phoneme classification

# training utterances	CNN L1	CRBM L1 [55]	MFCC [55]
100	55.4%	53.7%	**58.3%**
200	59.1%	56.7%	**61.5%**
500	63.0%	59.7%	**64.9%**
1000	65.0%	61.6%	**67.2%**
2000	66.8%	63.1%	**69.2%**
3696	67.8%	64.4%	**70.8%**

Our results were obtained using the maximum summary statistic. Overall, as shown in Table 4.3, our representation performs better than the CRBM [55] for all numbers of training utterances (the variance was negligible). However, both our representation and [55] do not outperform MFCC based features.

4.6 Summary

We started the chapter by detailing the model settings and training set that we used for unsupervised learning from speech data using the CNN building block and PSD training algorithm. We compared our model against Lee et al.'s CRBM results and the MFCC baseline presented in [55] as well as performing a qualitative analysis of our model's learned features.

Through this qualitative analysis we found that our first layer weights were similar to those of the CRBM and consisted of harmonic pattern detectors and formant energy detectors, although the CRBM seems to show the trajectories of formants more so than ours.

The key difference between the CRBM and our CNN features is in the second layer representation. The CRBM second layer features showed differences for male and female speakers, whilst ours did not. This was consistent with the results for the gender classification task where our L1 features performed better than our L2 features whilst the opposite was true for the CRBM.

In the speaker identification experiment, the CRBM outperformed our model. This is most likely due the patch based training used in PSD compared to whole utterance learning in the CRBM. Despite this, our model still outperformed the baseline MFCC representation. In the final phoneme classification experiment, both our model and the CRBM however, did not outperform the baseline MFCC representation.

Chapter 5

Discussion

In this chapter, we will discuss the results found in our experimental work, compare our model with the current state-of-the-art techniques used in speech recognition and discuss some of the limitations of this study.

5.1 CRBM Comparison

Our deep CNN approach in general results in comparable performance against the CRBM deep architecture in [55] as seen in the previous chapter. In the gender classification results (Table 4.1) and speaker identification results (Table 4.2), we saw that our first layer representation outperforms our second layer representation but the opposite was true for the CRBM. We have also seen in the visualization of the learned weights in Fig. 4.7 that the CRBM maintains a difference in the encoding of male and female speech in the second layer whilst our model does not. All this suggests that for our PSD trained CNN model, higher-level representations are likely to become more and more invariant towards speaker characteristics compared to the CRBM.

In the phoneme classification results (Table 4.3), although our model outperforms the CRBM, neither model is better than the baseline MFCC representation. We believe that there are two reasons for this.

First is the lack of depth in the representation. As both models use convolution in time along with max-pooling, as the layers increase, so to does the temporal duration of each feature. The first layer features have a temporal duration of approximately 70ms and the second layer 270ms. A third layer would be on the order of seconds. For this reason, only the first layer representation was used in the phoneme classification experiments as this fits well with temporal duration of phonemes.

However, since Lee et al.'s work [55] in 2009 and this work in 2011 [26], deep learning has been used very effectively in speech recognition systems [32]. Key to their success has been the depth of the network [60] [92] which both the CRBM and our CNN lack. The models reviewed in [32] mostly use conventional fully connected deep architectures and one possible solution to our problem of a lack of depth is to build a fully connected deep architecture on top of the CNN architecture layer with the appropriate temporal duration for the task.

Secondly, both the CNN and CRBM architectures are trained using unsupervised learning with the goal of [55] to learn a generic audio representation. For best recognition performance, it would appear that unsupervised learning alone is insufficient and that supervised learning is also required as in the more conventional deep learning training strategies adopted by the models reviewed in [32].

5.2 Related Work

As mentioned previously, one possible use of our CNN model would be to use it in the lower layers of a deep neural network with fully connected layers built on top. This network could then be fine-tuned using supervised learning as part of a hybrid DNN/HMM speech recognition system. Such a system could work well in low-resource conditions where training data is limited and unsupervised pre-training is helpful [32].

However, when labelled training data is plentiful, the effectiveness of unsupervised pre-training is reduced in hybrid DNN/HMM systems [91]. Additionally, under these conditions, it is possible even to use layer-wise discriminative pre-training as opposed to unsupervised pre-training [77].

Using our model as a feature extractor for a speech recognition system has parallels with the TANDEM framework [28] where neural network outputs are used as features for a GMM/HMM speech recognition system. A key difference however, is that in TANDEM the neural network is trained supervised to classify phone states and the log posterior probabilities obtained from the neural network are used as the features to train the GMM.

An alternative to using posterior probabilities as features for GMMs in the TANDEM framework, is to use low dimensional, bottleneck features obtained from a bottleneck hidden layer in the neural network [23]. As discussed previously in Sec. 2.5.2, in [71], bottleneck features were obtained using a bottleneck autoencoder trained on

the output of a deep neural network. It is thought that the underlying structure of the processes responsible for generating speech is low dimensional [32], hence the use of bottleneck features. In our model, instead of using a bottleneck layer, we use an over-complete representation with a sparsity constraint to provide what is effectively a low dimensional representation.

What is described as a "convolutive" approach to obtaining bottleneck features was proposed in [87]. Here two neural networks with bottleneck layers are used. The first network is trained as per usual to classify phone states using a context window of 11 frames as input. The second network takes as input, the bottleneck features from the first network using input windows with offsets of -10, -5, 0, +5 and +10 frames from the central frame of the input window to be classified. To allow for joint training of the two networks, the first network is replicated five times, each for the five input offsets used by the second fully connected network.

By replicating the first network five times, there is a degree of weight sharing along with local receptive fields as each network takes as input a small window of frames. This can be seen as convolution in time similar to our model. However, the goal of convolution in [87] is different to ours. In [87], the aim is to provide extra context with the addition of offset features as input to the second network to aid in classifying the central window. Convolution provides a means to do so easily and to train both networks jointly. Our goal with convolution is to provide shift invariant feature detection along the time axis. This is achieved through a single frame shift when using convolution and with max-pooling over neighbouring convolutional units.

An early example of convolutional networks used with speech recognition is the Time-Delay Neural Network (TDNN) [48]. Similar to our model, convolution in time is used but there is no pooling layer. The TDNN consisted of one convolutional hidden layer and a convolutional output layer used to classify spoken word segments. The TDNN was proposed in 1990 just as continuous density HMM models were first being used for speech recognition. Since then, HMMs for speech recognition have proven more popular and their ability to model the temporal sequences in speech makes using convolution in time somewhat redundant.

An alternative to using convolution in time proposed in [2] is to use convolution along the frequency axis. This helps to take into account of speaker variabilities, some of which can manifest itself as shifts in formant frequencies which convolution and pooling can provide invariance towards. It can also provides some robustness to noise if noise is limited to a local part of the spectrum.

Additionally, as speech features in lower frequency bands are very different to those in higher frequency bands, [2] also proposes to limit weight sharing to units within a max-pooling group only. This limited weight sharing however has the disadvantage that no further convolutional layers can be added on top as features are no longer related. Instead, fully connected layers need to be used to provide further depth. For our model, we faced a similar problem in that different feature maps are not related and thus how to perform convolution using these disparate feature maps as input to the next convolutional layer. To get around this, our model uses convolution kernels that extends over all of the feature maps together in the previous layer (although still localized in time) which can thought of as a fully connected style of approach.

Initial experiments using convolution in frequency and limited weight sharing under a hybrid CNN/HMM framework on the TIMIT dataset shows that the CNN/HMM model outperforms a DNN/HMM with a similar number of parameters with max-pooling being a critical factor in improving phoneme error rate [2]. In follow-up work [1], the CNN was pre-trained using the CRBM method and was found to give a performance boost over no pre-training on a large vocabulary, 18 hour Microsoft internal voice search dataset.

In addition to convolution in frequency, [1] also investigated the use of convolution in time. They found that convolution in time was not as effective as convolution in frequency as pooling can cause difficulty for higher layers to effectively label the central frame. Using both convolution in frequency and time gives roughly the same performance as full weight sharing in the frequency domain without convolution in time. Again, limited weight sharing in the frequency domain proved to be the most effective. Finally [1] also proposed a weighted softmax pooling method in order to try and learn the pooling size and structure but this did not prove effective.

Convolution in frequency using both hybrid CNN/HMM and TANDEM frameworks were investigated in [72] and [70] on the much larger scale 430 hour Broadcast News and 300 hour Switchboard datasets. Results again show the effectiveness of using a convolutional approach in the frequency domain on these larger tasks. In [72], convolution in time was also used but without max-pooling and in [70] using pooling in time did not prove effective. In both cases, no unsupervised pre-training was used.

Finally in this section, we will cover the recently proposed maxout network [22] which has set the state-of-the-art in various computer vision benchmarks. In the maxout network, instead of using a conventional non-linearity such as a sigmoid or tanh

function, a maxout unit takes the maximum of a group of linear input units. This allows the maxout unit to learn the appropriate activation function as it is parameterised by weights of its linear input units. A maxout unit can learn a piecewise linear approximation of any convex function and a maxout network is a universal approximator [22].

Maxout units perform the same function as max-pooling over linear units in a fully connected neural network. In convolutional neural networks, maxout units pool over units in different feature maps whilst max-pooling pools over units within the same feature map in order to achieve shift invariance. In [22], a convolutional maxout layer is followed by a regular max-pooling layer.

Initial work applying maxout networks to speech recognition has shown maxout to be effective under low resource conditions but gains are less pronounced in larger tasks where more training data is available [9] [56] [84]. Maxout networks were found to converge much faster than sigmoid non-linearities due to their facilitation of larger gradients in lower layers, alleviating the vanishing gradient problem [9] [84]. All work so far has focused only on fully connected networks and do not use convolutional methods.

5.3 Limitations of this Study

We recognise that this study is by no means a comprehensive evaluation. We have only evaluated one particular size of architecture in order to be as comparable as possible with the CRBM results in [55]. This study also only uses the TIMIT dataset which is a clean speech dataset recorded in a noise-free environment. It does not evaluate the representation's handling of noise and mismatched channel conditions, something that all good speech representations need to overcome. The experiments chosen to evaluate the representation were simplified versions of real world tasks such as phoneme recognition and open-set speaker verification. Nevertheless, this study is still worthwhile as a feasibility study into the use of convolutional architectures for the speech domain.

5.4 Summary

We have seen that our model performs comparably to the CRBM model under the three tasks performed. However, on the phoneme classification task, neither our model nor the CRBM performs better than the MFCC baseline. We believe that with added depth

and supervised learning, we can bridge that performance gap.

In the review of related work in speech recognition, our work in creating a feature extractor using a convolutional neural network has parallels with the TANDEM framework for speech recognition but the biggest point of difference is the use of supervised learning in TANDEM compared to our use of unsupervised learning only.

Additionally, we have seen that our use of convolution in time may not be an effective strategy as temporal modelling can be handled well with Hidden Markov Models. A more effective strategy could be to use convolution in the frequency domain. Convolutional maxout layers could also provide a useful compliment to max-pooling within feature maps though this has yet to be investigated in the speech domain.

Finally, we addressed some of the limitations of this study, including the use of clean data only, the use of simplified tasks and no exploration of different network architectures.

Chapter 6

Conclusion

The overall aim of this work was to carry out a feasibility study into the use of deep Convolutional Neural Networks (CNN) in the speech domain. CNNs were designed based on models of the visual cortex [51] and have been very effective in learning hierarchical representations for object recognition (Sec. 2.5.3). As there are many parallels between the visual and auditory cortex, we believed that applying CNNs to the speech domain may yield similar hierarchical representations.

At the time of this work, deep learning was starting to be applied in the speech domain. Most work was based on the Restricted Boltzmann Machine (RBM) architecture and a second aim of this study was to determine if an alternative building block to the RBM could be successfully used in the speech domain.

To achieve our goals, we developed a CNN building block trained unsupervised using the Predictive Sparse Decomposition algorithm simply modified for use with speech data (Chapter 3). We followed the experimental protocol and compared our model against Lee et al.'s work [55] which used a convolutional RBM (CRBM) as the building block for a two layer network applied to gender classification, speaker identification and phoneme classification (Chapter 4). We found that our CNN model performed comparably on these tasks with the CRBM showing that an alternative to RBM-based architectures can be used in the speech domain.

We also performed a qualitative comparison of the features learned by our CNN model and the CRBM. We found that in the first layer the features learned consisted of harmonic pattern detectors and formant energy detectors, similar to those learned by the CRBM, although the CRBM features showed more obvious formant trajectories than our features did. We found that for the second layer, the features learned were different. In a visualization showing which features were most active on female and male

speech (Fig. 4.7), the CRBM retained a different encoding in the second layer whilst our model did not. Results on the gender classification and speaker identification tasks showed that our first layer representation outperformed the second layer representation whilst the opposite was true for the CRBM. This suggests that our model becomes more invariant towards speaker characteristics as we move up the feature hierarchy compared to the CRBM.

When compared against the baseline MFCC representation, in particular on the phoneme classification task (Table 4.3), both our CNN model and the CRBM model performed poorly by comparison. We have seen in other work that deep networks have been very effective when used as acoustic models in speech recognition systems [32]. We hypothesize that the reasons for the convolutional models' poor performance was due to a lack of depth in the representation and a lack of supervised training which we will need to explore in future work.

Overall, this study has shown that using the Predictive Sparse Decomposition algorithm to learn the weights of a deep CNN unsupervised can be an alternative to using the Convolutional Restricted Boltzmann Machine.

6.1 Future Work

In this feasibility study we have focused only on one network architecture. In order to better understand the characteristics of our model, we would need to explore the architecture space to find the optimal number of layers, number of feature maps per layer, size of convolution kernels and number of units to pool over. As neural networks do not need decorrelated input, using PCA whitening to transform the input spectrogram may not be necessary. Investigating the use of the raw spectrogram and other types of input features may also prove fruitful.

We will also need to run further experiments to test our hypotheses for the poor performance on phoneme classification by testing the effectiveness of using supervised fine-tuning and the effectiveness of adding fully connected layers to increase the depth of the network. Additionally, increasing depth will also help to us to investigate the hypothesis that our model becomes more speaker invariant with depth.

Finally, in light of recent results using convolution in the frequency domain for speech recognition [2] [1] [72] [70], it would be prudent to investigate whether we can also apply PSD with convolution in the frequency domain and whether or not it will prove to be more effective than convolution in time.

Bibliography

[1] Ossama Abdel-Hamid, Li Deng, and Dong Yu. Exploring convolutional neural network structures and optimization techniques for speech recognition. In *Proceedings of the 14th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, pages 3366–3370, 2013.

[2] Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, and Gerald Penn. Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 4277–4280, 2012.

[3] Nasir Ahmed, T. Natarajan, and Kamisetty Ramamohan Rao. Discrete cosine transform. *IEEE Transactions on Computers*, C–23(1):90–93, Jan 1974.

[4] Pierre Baldi and Kurt Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, 2(1):53–58, 1989.

[5] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems (NIPS) 19*, pages 153–160, 2006.

[6] Christopher M. Bishop. *Neural Networks for Pattern Recognition*, pages 116–160, 195–200, 230–240. Oxford University Press, 1995.

[7] Herve Bourlard and Y. Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59(4-5):291–294, 1988.

[8] Mike Brookes. VOICEBOX: Speech processing toolbox for MATLAB. `http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html`. [Online: accessed 01/10/2010].

[9] Meng Cai, Yongzhe Shi, and Jia Liu. Deep maxout neural networks for speech recognition. In *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 291–296, 2013.

[10] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

[11] Ke Chen and Ahmad Salman. Extracting speaker-specific information with a regularized siamese deep network. In *Advances in Neural Information Processing Systems (NIPS) 24*, pages 298–306, 2011.

[12] Steven B. Davis and Paul Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentence. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(4):357–366, 1980.

[13] David DeMers and Garrison W. Cottrell. Non-linear dimensionality reduction. In *Advances in Neural Information Processing Systems (NIPS) 5*, pages 580–587, 1992.

[14] Li Deng, Michael Seltzer, Dong Yu, Alex Acero, Abdel-rahman Mohamed, and Geoffrey E. Hinton. Binary coding of speech spectrograms using a deep auto-encoder. In *Proceedings of the Eleventh Annual Conference of the International Speech Communication Association (INTERSPEECH)*, pages 1692–1695, Sep 2010.

[15] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11:625–660, 2010.

[16] Dumitru Erhan, Pierre-Antoine Manzagol, Yoshua Bengio, Samy Bengio, and Pascal Vincent. The difficulty of training deep architectures and the effect of unsupervised pre-training. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 153–160, 2009.

[17] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980.

[18] John S. Garofolo, Lori F. Lamel, William M. Fisher, Jonathan G. Fiscus, David S. Pallett, Nancy L. Dahlgren, and Victor Zue. *TIMIT Acoustic-Phonetic Continuous Speech Corpus*. Linguistic Data Consortium, Philadelphia, 1993.

[19] Jort F. Gemmeke and Bert Cranen. Sparse imputation for noise robust speech recognition using soft masks. In *Proceedings of the IEEE Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4645–4648, 2009.

[20] Jort F. Gemmeke, Ulpu Remes, and Kalle J. Palomaki. Observation uncertainty measures for sparse imputation. In *Proceedings of the Eleventh Annual Conference of the International Speech Communication Association (INTERSPEECH)*, pages 2262–2265, Sep 2010.

[21] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 315–323, 2010.

[22] Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron C. Courville, and Yoshua Bengio. Maxout networks. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, pages 1319–1327, 2013.

[23] František Grézl, Martin Karafiát, Stanislav Kontár, and Jan Černocký. Probabilistic and bottle-neck features for LVCSR of meetings. In *Proceedings of the IEEE Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 757–760, 2007.

[24] Roger Grosse, Rajat Raina, Helen Kwong, and Andrew Y. Ng. Shift-invariant sparse coding for audio classification. In *Proceedings of the Twenty-third Conference on Uncertainty in Artificial Intelligence*, pages 149–158, 2007.

[25] Raia Hadsell, Ayse Erkan, Pierre Sermanet, Marco Scoffier, and Urs Muller. Deep belief net learning in a long-range vision system for autonomous off-road driving. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 628–633, 2008.

[26] Darren Hau and Ke Chen. Exploring hierarchical speech representations with a deep convolutional neural network. In *Proceedings of the 11th UK Workshop on Computational Intelligence*, pages 37–42, 2011.

[27] Mikael Henaff, Kevin Jarrett, Koray Kavukcuoglu, and Yann LeCun. Unsupervised learning of sparse features for scalable audio classification. In *International Society for Music Information Retrieval Conference*, pages 681–686, 2011.

[28] Hynek Hermanksy, Daniel P. W. Ellis, and Sangita Sharma. Tandem connectionist feature extraction for conventional HMM systems. In *Proceedings of the IEEE Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1635–1638, 2000.

[29] Yasser Hifny and Steve Renals. Speech recognition using augmented conditional random fields. *IEEE Transactions on Audio, Speech & Language Processing*, 17(2):354–365, 2009.

[30] Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002.

[31] Geoffrey E. Hinton. A practical guide to training restricted boltzmann machines. Technical report, University of Toronto, 2010.

[32] Geoffrey E. Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[33] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–54, 2006.

[34] Geoffrey E. Hinton and Ruslan R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–7, 2006.

[35] Guoning Hu and DeLiang Wang. Auditory segmentation based on onset and offset analysis. *IEEE Transactions on Audio, Speech and Language Processing*, 15(2):396–405, 2007.

[36] Xeudong Huang, Alex Acero, and Hsiao-Wuen Hon. *Spoken Language Processing: A Guide to Theory, Algorithm and System Development*, pages 21–36, 276–283, 464–468. Prentice Hall, 2001.

[37] Xuedong Huang, Alex Acero, Fileno A. Alleva, Li Jiang, Mei-Yuh Hwang, and Milind Mahajan. From Sphinx II to Whisper – Making Speech Recognition Usable. In Chin-Hui Lee, Frank K. Soong, and Kuldip K. Paliwal, editors, *Automatic Speech Recognition Speech and Speaker Recognition*, pages 481–508. Norwell, MA, Kluwer Academic Publishers, 1996.

[38] David H. Hubel and Torsten N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160(1):106–154, 1962.

[39] Anthony F. Jahn and Joseph Santos-Sacchi. *Physiology of the Ear*, pages 549–569, 613–632, 651–667. Singular Thomson Learning, second edition, 2001.

[40] Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *Proceedings of the 12th International Conference on Computer Vision (ICCV)*, pages 2146–2153, 2009.

[41] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*, pages 207–246, 319–367. Pearson Education, Inc., second edition, 2009.

[42] Koray Kavukcuoglu, Marc'Aurelio Ranzato, Rob Fergus, and Yann LeCun. Learning invariant features through topographic filter maps. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1605–1612, 2009.

[43] Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. Fast inference in sparse coding algorithms with applications to object recognition. Technical report, Computational and Biological Learning Lab, Courant Institute, New York University, 2008. CBLL-TR-2008-12-01.

[44] Koray Kavukcuoglu, Pierre Sermanet, Y-Lan Boureau, Karol Gregor, Michael Mathieu, and Yann LeCun. Learning convolutional feature hierarchies for visual recognition. In *Advances in Neural Information Processing Systems (NIPS) 23*, pages 1090–1098, 2010.

[45] T. Kinnunen, E. Karpov, and P. Franti. Real-time speaker identification and verification. *IEEE Transactions on Audio, Speech and Language Processing*, 14(1):277–288, 2006.

[46] Alex Krizhevsky and Geoffrey E. Hinton. Using very deep autoencoders for content-based image retrieval. In *19th European Symposium on Artificial Neural Networks (ESANN)*, 2011.

[47] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS) 25*, pages 1106–1114, 2012.

[48] Kevin J. Lang, Alex Waibel, and Geoffrey E. Hinton. A time-delay neural network architecture for isolated word recognition. *Neural Networks*, 3(1):23–43, 1990.

[49] Hugo Larochelle, Yoshua Bengio, Jerome Louradour, and Pascal Lambin. Exploring strategies for training deep neural networks. *Journal of Machine Learning Research*, 10:1–40, 2009.

[50] Yann LeCun, Bernard Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne Hubbard, and Lawrence D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[51] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[52] Yann LeCun, Fu Jie Huang, and Léon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, pages 97–104, 2004.

[53] Honglak Lee, Chaitu Ekanadham, and Andrew Y. Ng. Sparse deep belief net model for visual area v2. In *Advances in Neural Information Processing (NIPS) 20*, 2007.

[54] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)*, pages 609–616, 2009.

[55] Honglak Lee, Yan Largman, Peter Pham, and Andrew Y. Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks.

In *Advances in Neural Information Processing Systems (NIPS) 22*, pages 1096–1104, 2009.

[56] Yajie Miao, Florian Metze, and Shourabh Rawat. Deep maxout networks for low-resource speech recognition. In *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 398–403, 2013.

[57] Ji Ming and F. Jack Smith. Improved phone recognition using bayesian triphone models. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 409–412, 1998.

[58] Vivienne L. Ming and Lori L. Holt. Efficient coding in human auditory perception. *Journal of the Acoustical Society of America*, 126(3):1312–20, 2009.

[59] Abdel-rahman Mohamed, George E. Dahl, and Geoffrey E. Hinton. Acoustic modeling using deep belief networks. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1):14–22, 2012.

[60] Abdel-rahman Mohamed, Geoffrey E. Hinton, and Gerald Penn. Understanding how deep belief networks perform acoustic modelling. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 4273–4276, 2012.

[61] Brian C. J. Moore. *Introduction to the Psychology of Hearing*, pages 21–52, 65–78. Academic Press, fifth edition, 2003.

[62] Bruno A. Olshausen and David J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

[63] Encyclopaedia Britannica Online. Basilar membrane: analysis of sound frequencies. [ART]. `http://www.britannica.com/EBchecked/media/537/The-analysis-of-sound-frequencies-by-the-basilar-membrane`. [Online: accessed 27/11/2013].

[64] Rajat Raina, Alexis Battle, Honglak Lee, Benjamin Packer, and Andrew Y. Ng. Self-taught learning: Transfer learning from unlabelled data. In *Proceedings of the 24th Annual International Conference on Machine Learning (ICML)*, pages 759–766, 2007.

[65] Marc'Aurelio Ranzato, Fu Jie Huang, Y-Lan Boureau, and Yann LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, 2007.

[66] Sourabh Ravindran, David V. Anderson, and Malcolm Slaney. Improving the noise-robustness of mel-frequency cepstral coefficients for speech processing. In *Statistical and Perceptual Audio Processing*, pages 48–52, 2006.

[67] Douglas A. Reynolds, Thomas F. Quatieri, and Robert B. Dunn. Speaker verification using adapted gaussian mixture models. *Digital Signal Processing*, 10(1-3):19–41, 2000.

[68] Maximilian Riesenhuber and Tomaso Poggio. Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2(11):1019–1025, 1999.

[69] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, pages 833–840, 2011.

[70] Tara N. Sainath, Brian Kingsbury, Abdel-rahman Mohamed, George E. Dahl, George Saon, Hagen Soltau, Tomás Beran, Aleksandr Y. Aravkin, and Bhuvana Ramabhadran. Improvements to deep convolutional neural networks for LVCSR. In *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 315–320, 2013.

[71] Tara N. Sainath, Brian Kingsbury, and Bhuvana Ramabhadran. Auto–encoder bottleneck features using deep belief networks. In *Proceedings of the IEEE Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4153–4156, 2012.

[72] Tara N. Sainath, Abdel-rahman Mohamed, Brian Kingsbury, and Bhuvana Ramabhadran. Deep convolutional neural networks for LVCSR. In *Proceedings of the IEEE Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8614–8618, 2013.

[73] Tara N. Sainath, Bhuvana Ramabhadran, David Nahamoo, Dimitri Kanevsky, and Abhinav Sethy. Exemplar-based sparse representation features for speech

recognition. In *Proceedings of the Eleventh Annual Conference of the International Speech Communication Association (INTERSPEECH)*, pages 2258–2261, Sep 2010.

[74] Ahmad Salman. *Learning speaker-specific characteristics with deep neural architecture*. PhD thesis, University of Manchester, 2012.

[75] Eric Saund. Dimensionality-reduction using connectionist networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(3):304–314, 1989.

[76] Andrew Saxe, Pang Wei Koh, Zhenghao Chen, Maneesh Bhand, Bipin Suresh, and Andrew Y. Ng. On random weights and unsupervised feature learning. In *Proceedings of the 28th Annual International Conference on Machine Learning (ICML)*, pages 1089–1096, 2011.

[77] Frank Seide, Gang Li, Xie Chen, and Dong Yu. Feature engineering in context-dependent deep neural networks for conversational speech transcription. In *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 24–29, 2011.

[78] Shihab Shamma. On the role of space and time in auditory processing. *Trends in Cognitive Sciences*, 5(8):340–348, 2001.

[79] Gariemella S. V. S. Sivaram, Sriram Ganapathy, and Hynek Hermansky. Sparse auto-associative neural networks: theory and application to speech recognition. In *Proceedings of the Eleventh Annual Conference of the International Speech Communication Association (INTERSPEECH)*, pages 2270–2273, Sep 2010.

[80] Malcolm Slaney and Richard F. Lyon. On the importance of time - a temporal representation of sound. In *Visual Representations of Speech Signals*, pages 95–116. John Wiley & Sons, Inc., New York, NY, USA, 1993.

[81] Paris Smaragdis. *Redundancy Reduction for Computational Audition, a Unifying Approach*. PhD thesis, Massachusetts Institute of Technology, June 2001. pages 41–45.

[82] Evan C. Smith and Michael S. Lewicki. Learning efficient auditory codes using spikes predicts cochlear filters. In *Advances in Neural Information Processing Systems (NIPS) 17*, pages 1289–1296, 2004.

[83] Pawel Swietojanski, Arnab Ghosal, and Steve Renals. Unsupervised cross-lingual knowledge transfer in DNN-based LVCSR. In *Proceedings of the IEEE Spoken Language Technology Workshop (SLT)*, pages 246–251, 2012.

[84] Pawel Swietojanski, Jinyu Li, and Jui-Tang Huang. Investigation of maxout neural networks for speech recognition. In *Proceedings of the IEEE Conference on Acoustics, Speech and Signal Processing (ICASSP) (to appear)*, 2014.

[85] Graham W. Taylor, Geoffrey E. Hinton, and Sam Roweis. Modeling human motion using binary latent variables. In *Advances in Neural Information Processing Systems (NIPS) 19*, pages 1345–1352, 2006.

[86] Vivek Tyagi and Christian Wellekens. On desensitizing the mel-cepstrum to spurious spectral components for robust speech recognition. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 529–532, 2005.

[87] Karel Veselý, Martin Karafiát, and František Grézl. Convolutive bottleneck network features for LVCSR. In *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 42–47, 2011.

[88] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine learning (ICML)*, pages 1096–1103, 2008.

[89] Wikipedia. Speech production. `https://en.wikipedia.org/wiki/Speech_production`. [Online: accessed 26/11/2013].

[90] Philip C. Woodland, Chris J. Leggetter, J. J. Odell, V. Valtchev, and Steve J. Young. The 1994 HTK large vocabulary speech recognition system. In *Proceedings of the IEEE Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 73–76, 1995.

[91] Dong Yu, Li Deng, and George E. Dahl. Roles of pretraining and fine-tuning in context-dependent DBN-HMMs for real-world speech recognition. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2010.

[92] Dong Yu, Michael L. Seltzer, Jinyu Li, Jui-Ting Huang, and Frank Seide. Feature learning in deep neural networks - studies on speech recognition tasks. In *International Conference on Learning Representations*, 2013.

[93] Xiaojia Zhao and DeLiang Wang. Analyzing noise robustness of MFCC and GFCC features in speaker identification. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 7204–7208, 2013.

Appendix A

Implementation

The experiments presented in this thesis were implemented using MATLAB. Reading TIMIT speech files and extracting Mel-frequency Cepstral Coefficients were performed using the VOICEBOX toolbox [8]. The LibSVM library [10] was used for the Support Vector Machine implementation. All other code was written by the author of this thesis. Below follows a listing of the source code used to implement the Predictive Sparse Decomposition training algorithm.

First, psd_patch.m which is the main training function.

```
function [ weights biases gain basis mse ] = psd_patch( train, valid, ...
    numCodeUnits, numEpochs, basisLearningRate, predictorLearningRate, ...
    decayLearningRate, codeSparsity, predictionAlpha, codeOptThreshold, ...
    codeOptLearningRate, codeOptMaxIterations, codeLRSearchMaxIterations, ...
    loggingRate )
%PSD_PATCH Run PSD using dataset consisting of vectorized patches
%  Input args:
%    - train/valid: Training and validation set consisting of vectorized
%    patches. Vectorized patch dim x num data points
%    - numCodeUnits: Number of hidden units
%    - numEpochs: Number of epochs to train for
%    - basisLearningRate: Learning rate for basis (decoder) part
%    - predictorLearningRate: Learning rate for predictor (encoder) part
%    - decayLearningRate: Learning rate decay factor
%    - codeSparsity: Weighting parameter of the sparsity part of the loss
%    function
%    - predictionAlpha: Weighting parameter of the predictor part of the
%    loss function
%    - codeOptThreshold: Convergence threshold for stopping search for
%    optimal sparse code
%    - codeOptLearningRate: Learning rate for searching for optimal sparse
%    code
%    - codeOptMaxIterations: Maximum number of iterations for searching for
%    optimal sparse code
```

```
%     - codeLRSearchMaxIterations: Maximum number of learning rate search
%     iterations per iteration of searching for optimal sparse code
%     - loggingRate: Rate at which to compute error and decay learning rate
%     (number of data points)
%  Output:
%     - weights: Predictor weights, num code units x vectorized patch dim
%     - biases: Predictor biases, num code units x 1
%     - gain: Predictor gains, num code units x 1
%     - basis: Decoder basis, vectorized patch dim x num code units

%% Initialize parameters

fprintf(1,'Initializing parameters...');

[inputDim numPatches] = size(train);

% Randomly initialize basis functions according to uniform distribution
% [-1/sqrt(fan-in), 1/sqrt(fan-in)].
upper = numCodeUnits^(-0.5);
lower = -upper;
basis = lower + (upper - lower).*rand(inputDim, numCodeUnits);
% Normalize each basis function (column)
for j = 1:numCodeUnits
 basis(:,j) = basis(:,j)./norm(basis(:,j));
end

% Randomly initialize basis functions according to uniform distribution
% [-1/sqrt(fan-in), 1/sqrt(fan-in)].
upper = inputDim^(-0.5);
lower = -upper;
weights = lower + (upper - lower).*rand(numCodeUnits, inputDim);

biases = zeros(numCodeUnits, 1);

gain = ones(numCodeUnits, 1);

zeroThreshold = 0.0000001;

basisLearningRateCurrent = basisLearningRate;
predictorLearningRateCurrent = predictorLearningRate;

fprintf(1,'Completed\r');

%% Train

fprintf(1,strcat('[',datestr(now),'] '));
fprintf(1,' Calculating initial mse...\r');
mse = zeros((numPatches*numEpochs)/loggingRate + 1, 4);
[ mseTrain mseValid ] = calc_train_valid_mse( train, valid, basis, weights, ...
    biases, gain, codeSparsity );
mse(1,1) = 0;
mse(1,2) = mseTrain;
mse(1,3) = mseValid;
fprintf(1,strcat('[',datestr(now),'] '));
```

```
fprintf(' Train = %f, Valid = %f\r', mseTrain, mseValid);

for epoch=1:numEpochs
    fprintf(1,strcat('[',datestr(now),'] '));
    fprintf(1,' Starting epoch %i of %i...\r', epoch, numEpochs);
    sumOptimizeIterations = 0;
    for p=1:numPatches
        % Predict the code vector Z
        patch = train(:,p);
        nonLinearOutput = tanh(weights*patch + biases);
        predictedCode = gain.*nonLinearOutput;

        % Optimize the code vector to find Z*
        [optimalCode numIterations] = optimize_z(patch, predictedCode, basis, ...
            codeSparsity, predictionAlpha, codeOptThreshold, ...
            codeOptLearningRate, codeOptMaxIterations, ...
            codeLRSearchMaxIterations, zeroThreshold);
        sumOptimizeIterations = sumOptimizeIterations + numIterations;

        % Update basis functions
        derivBasis = (patch - basis*optimalCode);
        derivBasis = derivBasis * (-optimalCode)';
        basis = basis - basisLearningRateCurrent * derivBasis;
        % Normalize each basis function (column)
        for j = 1:numCodeUnits
            basis(:,j) = basis(:,j)./norm(basis(:,j));
        end

        % Update weights and biases
        codeDeriv = (1./gain) .* (gain.^2 - predictedCode.^2);
        delta = (-codeDeriv).*(optimalCode - predictedCode);
        delta = predictionAlpha * delta;
        biases = biases - predictorLearningRateCurrent * delta;
        derivWeights = delta * patch';
        weights = weights - predictorLearningRateCurrent * derivWeights;
        % Update gain
        derivGain = predictionAlpha*(optimalCode - ...
            predictedCode).*(-nonLinearOutput);
        gain = gain - predictorLearningRate * derivGain;

        if(mod(p,loggingRate)==0)
            % Learning rate decay
            predictorLearningRateCurrent = predictorLearningRate / (( ...
                (((epoch-1)*numPatches)+p) /1000 * decayLearningRate) + 1);
            basisLearningRateCurrent = basisLearningRate / (( ...
                (((epoch-1)*numPatches)+p) /1000 * decayLearningRate) + 1);

            fprintf(1,strcat('[',datestr(now),'] '));
            fprintf(1,' Completed working on patch %i of %i (E%i)\r', p, ...
                numPatches, epoch);
            [ mseTrain mseValid ] = calc_train_valid_mse( train, valid, basis, ...
                weights, biases, gain, codeSparsity );
            patchNum = numPatches*(numEpochs - 1) + p;
            idx = patchNum / loggingRate + 1;
```

```
            mse(idx, 1) = patchNum;
            mse(idx, 2) = mseTrain;
            mse(idx, 3) = mseValid;
            fprintf(1,strcat('[',datestr(now),'] '));
            fprintf(' Train = %f, Valid = %f\r', mseTrain, mseValid);
        end
    end
    fprintf(1,strcat('[',datestr(now),'] '));
    fprintf(1,' Completed epoch %i of %i\r', epoch, numEpochs);
    mse(end, 4) = sumOptimizeIterations / numPatches;
end

fprintf(1,strcat('[',datestr(now),'] '));
fprintf(1,' Completed Training\r');

end
```

Next, optimize_z.m which is used to find the optimal sparse code called by psd_patch.m.

```
function [ optimalZ numOptimizeIterations ] = optimize_z( patch, predictedZ, ...
    basis, sparsityParam, predictionAlpha, convergenceThreshold, ...
    initialLearningRate, maxOptimizeIterations, maxLRSearchIterations, ...
    zeroThreshold )
%OPTIMIZE_Z Optimize the code vector z using gradient descent
%   Input args:
%    - patch: Original input patch as column vector
%    - predictedZ: Code (column) vector z from predictor
%    - basis: Matrix of basis functions of size patchLength * codeLength
%    - sparsityParam: Scalar to control sparsity of the z
%    - predictionAlpha: Scalar to control influence of prediction loss
%    - convergenceThreshold: Threshold used to determine convergence (scalar)
%    - learningRate: Learning rate (scalar)
%    - maxOptimizeIterations: Maximum number of iterations before stopping
%    - maxLRSearchIterations: Maximum number of learning rate search iterations
%    - zeroThreshold: Threshold used to determine floating point zero value
%   Output:
%    - optimalZ - Optimal code vector z*


z = predictedZ;

delta = convergenceThreshold + 1;
numOptimizeIterations = 0;
learningRate = initialLearningRate;
initialLoss = (norm(patch - basis*z)).^2 + (sparsityParam * norm(z,1));
currentLoss = initialLoss;
previousLoss = Inf;
while (delta > convergenceThreshold && currentLoss < previousLoss && ...
    numOptimizeIterations < maxOptimizeIterations)

    % Calculate derivatives
    decoderDeriv = (patch - basis*z);
```

```
    decoderDeriv = (-basis)' * decoderDeriv;
    predictionDeriv = predictionAlpha * (z - predictedZ);
    sparsityDeriv = sparsityParam * sign_float(z, zeroThreshold);
    deriv = decoderDeriv + predictionDeriv + sparsityDeriv;

    % Use search to find appropriate learning rate
    numLRSearchIterations = 0;
    newLoss = currentLoss + 1;

    % Keep decreasing the learning rate until the loss starts to decrease
    % or hit maximum number of search iterations
    while (newLoss > currentLoss && numLRSearchIterations < maxLRSearchIterations)
        % Take a trial gradient step
        tempZ = z - learningRate * deriv;
        % Calculate new loss with trial z
        newLoss = (norm(patch - basis*tempZ)).^2 + (sparsityParam * ...
            norm(tempZ,1)) + (predictionAlpha * (norm(tempZ - predictedZ).^2));
        % Increment number of search iterations
        numLRSearchIterations = numLRSearchIterations + 1;
        % Loss is decreasing, make the trial step permanent
        if (newLoss < currentLoss)
            % Update z
            z = tempZ;
            % Update losses
            previousLoss = currentLoss;
            currentLoss = newLoss;
        else
            % Halve the learning rate
            learningRate = learningRate / 2;
        end
    end

    % Calculate the L2 norm of the derivative
    delta = norm(deriv);

    numOptimizeIterations = numOptimizeIterations + 1;
    %fprintf(1,'Iteration %i: Derivative norm = %f\r',numIterations, delta);
end
optimalZ = z;
finalLoss = (norm(patch - basis*optimalZ)).^2 + (sparsityParam * ...
    norm(optimalZ,1)) + (predictionAlpha * (norm(optimalZ - predictedZ).^2));
%fprintf(1,'Optimal z found in %i iterations (%i search iterations). Initial ...
    Loss = %f, Final Loss = %f\r',numOptimizeIterations, ...
    numLRSearchIterations, initialLoss, finalLoss);

end
```

Finally, calc_train_valid_mse.m which is used to compute the mean squared reconstruction error of the training and validation datasets called by calc_train_valid_mse.m.

```
function [ meanTrainError meanValidError ] = calc_train_valid_mse( train, ...
    valid, basis, weights, biases, gain, codeSparsity )
%CALC_TRAIN_VALID_MSE Computes the reconstruction error for training and
%validation sets using the predictor for computing the representation
%  Input args:
%    - train/valid: Training and validation set consisting of vectorized
%    patches. Vectorized patch dim x num data points
%    - basis: Decoder basis, vectorized patch dim x num code units
%    - weights: Predictor weights, num code units x vectorized patch dim
%    - biases: Predictor biases, num code units x 1
%    - gain: Predictor gains, num code units x 1
%    - codeSparsity: Weighting parameter of the sparsity part of the loss
%    function
%  Output:
%    - meanTrainError: Mean squared reconstruction error of the training set
%    - meanValidError: Mean squared reconstruction error of the validation set


[inputDim numPatches] = size(train);
sumSquaredError = 0;
for i=1:numPatches
    % Predict the code vector Z
    patch = train(:,i);
    predictedCode = gain.*tanh(weights*patch + biases);
    patchSquaredError = sum((patch - basis*predictedCode).^2);
    sumSquaredError = sumSquaredError + patchSquaredError;
end
meanTrainError = sumSquaredError / numPatches;

[inputDim numPatches] = size(valid);
sumSquaredError = 0;
for i=1:numPatches
    % Predict the code vector Z
    patch = valid(:,i);
    predictedCode = gain.*tanh(weights*patch + biases);
    patchSquaredError = sum((patch - basis*predictedCode).^2);
    sumSquaredError = sumSquaredError + patchSquaredError;
end
meanValidError = sumSquaredError / numPatches;

end
```