Automatic Speech Feature Extraction using a Convolutional Restricted Boltzmann Machine

David John Anderson

A dissertation submitted to the Faculty of Science, University of the Witwatersrand, in fulfilment of the requirements for the degree of Master of Science
2017

Declaration

I declare that this thesis was composed by myself and that the work contained therein is my own, except where explicitly stated otherwise in the text.

(*David John Anderson*)

To my oldest friend C.J.F. Reyneke (IV). Thanks for the many late night Skype discussions which helped consolidate the techniques and outcomes in my mind.

The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.

Abstract

Restricted Boltzmann Machines (RBMs) are a statistical learning concept that can be interpreted as Artificial Neural Networks. They are capable of learning, in an unsupervised fashion, a set of features with which to describe a data set. Connected in series RBMs form a model called a Deep Belief Network (DBN), learning abstract feature combinations from lower layers. Convolutional RBMs (CRBMs) are a variation on the RBM architecture in which the learned features are kernels that are convolved across spatial portions of the input data to generate feature maps identifying if a feature is detected in a portion of the input data. Features extracted from speech audio data by a trained CRBM have recently been shown to compete with the state of the art for a number of speaker identification tasks. This project implements a similar CRBM architecture in order to verify previous work, as well as gain insight into Digital Signal Processing (DSP), Generative Graphical Models, unsupervised pre-training of Artificial Neural Networks, and Machine Learning classification tasks. The CRBM architecture is trained on the TIMIT speech corpus and the learned features verified by using them to train a linear classifier on tasks such as speaker genetic sex classification and speaker identification. The implementation is quantitatively proven to successfully learn and extract a useful feature representation for the given classification tasks.

Contents

Chapter 1

Introduction

Defining and extracting information from audio data is a difficult problem to solve due to the dimensionality of the data and the complexity of the information. Traditional approaches for doing so have taken many years of hard work and expertise to craft. While much progress has been made manually through the years, Machine Learning (ML) techniques are starting to prove a worthy alternative to the manual approach.

Classification tasks all follow a similar format: Define and extract 'features' in the data, represent these as a feature vector, and find a mapping function that maps the input feature vector to a particular class label. Learning a successful classification function will depend on the ability of the feature set to represent complex and abstract information in a manner that algorithms are easily able to separate into class categories [26]. In the case of audio and speech data, it makes sense to exploit an understanding of well-defined psycho-acoustic and musical properties to create reduced feature sets relevant to the human listener. This has resulted in developments such as Mel Frequency Cepstral Coefficients[1], loudness, pitch and harmonic features [49] as well as rhythmic content features [48].

The problem with manually crafting feature sets to use in classification tasks is that these properties take a lot of time and effort to define, inherently remove information from the data that we may not perceive as relevant but may be, and are not always suited to different classification tasks. This implies that the architect of the feature set has a rich and deep understanding of the data, which in audio signals requires a lot of study and expertise to grasp the necessary mathematical rigour. In addition to this, the aforementioned feature sets are lower-level than high-level abstract information we are often interested in such as musical genre, the emotional content of a speech sample, or which speaker out of a set speakers is talking.

It is due to this complexity, as well as advancements in ML, that researchers have been employing new promising techniques in the automatic extraction of features from

[1] Calculated by a transformation of a signal's power spectrum, resulting in a representation of useful portions of the speech amplitude spectrum in a compact form [34].

audio data. Lee et al. [32] present such a technique known as Convolutional Restricted Boltzmann Machines (CRBMs) and test their ability to extract useful features from audio samples. The aim of this project is to investigate and verify the work presented in Lee et al. [32]. This is done by re-implementing the CRBM layer presented by the authors, training it on the same data set, and performing two verification experiments using the learned features. The experiments verify the ability of the learned features to represent compact information in the speech data useful for classifying a speakers genetic sex, as well as identifying a given speaker out of a set of 168 unique speakers.

In order to provide context to the work done in this project, Chapter 2 outlines the core concepts that support the implementation. Chapter 3 describes the implementation details and a break down of the scope implemented for this project as well as third-party libraries used. Descriptions of the experiments performed are then introduced in Chapter 4. The subsequent chapters then present and discuss the experimental results, as well as insights learned through the course of the project.

Chapter 2

Background

2.1 Introduction

To understand the implementation and experiments that probe the problem at hand, a number of theoretical concepts and related work must be introduced and placed in context. At the very centre of the implementation is the Restricted Boltzmann Machine (RBM). Restricted Bolzmann Machines are Probabilistic Graphical Models that can be interpreted and treated as Neural Networks and are capable of creating reduced representations of their input data from which recreations of the original input can be made. To understand how this is achieved, knowledge of Probabilistic Graphical Models is required. Understanding Graphical Models and how to use them to perform statistical inference in turn requires knowledge of Bayesian probability theory. Furthermore, as the problem at hand deals with processing and operating on digital audio data, concepts around the data representation, processing and output are introduced first, along with Principal Component Analysis as a technique to reduce the overall dimensionality of the final input-data representation. The sections that follow then introduce the pre-requisite concepts behind RBMs by covering Neural Networks, the Bayesian Probability Model and Probabilistic Graphical Models. Once this is covered, the theory behind RBMs and convolutional RBMs is presented.

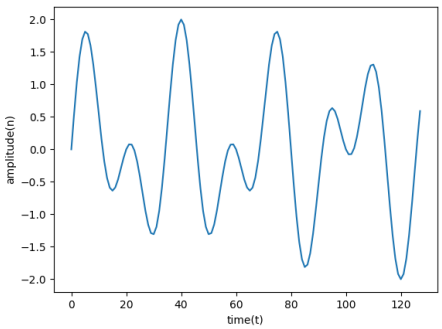
2.2 Audio Data Representation

In order to understand how to operate on audio data, one must first understand the data and how it is represented digitally. In the real world sound is a collection of continuous waves that propagate through the air. These pressure waves are heard via a biological process in the ear in which the waves cause the eardrum to vibrate following the waveform of the sound. These vibrations are converted into signals that are fed into the brain and interpreted as the sounds we hear [24].

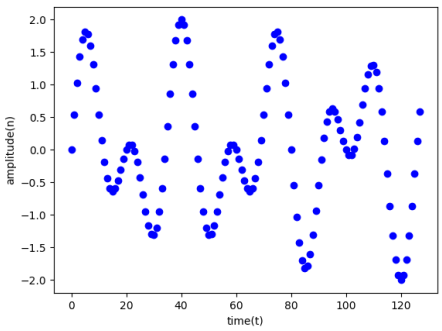
In order to represent these continuous waves digitally, they must be discretised. This process of discretisation, also known as analog-to-digital conversion, works by simply taking a discrete sample of the continuous wave a certain number of times per second. The number of times per second a sample is taken is also known as the sample rate. The difference between an analog and digital representation of a sound wave is illustrated in Figures 2.1a and 2.1b respectively. The choice of sample rate is directly affected by the highest frequency one wishes to capture during the analog to digital conversion. If the sample rate is too low, characteristics of the real wave can be lost or distorted due to an effect known as "aliasing", illustrated in Figure 2.2. In order to accurately capture a frequency, the sample rate must be approximately double that of the frequency [6]. This is a direct consequence of the Nyquist-Shannon sampling theorem which states:

Theorem 1. *If a function $x(t)$ contains no frequencies higher than B hertz, it is completely determined by giving its ordinates at a series of points spaced $\frac{1}{2B}$ seconds apart. [42]*

We can now mathematically represent a digital signal as a single vector $\boldsymbol{x}$ where each element $\boldsymbol{x}[t]$ is the pressure sample at time-step t, and $t \in \mathbb{N}$.



(a) Continuous wave - 1kHz sine + 1.8kHz sine



(b) Discrete wave - 1kHz sine + 1.8kHz sine

Figure 2.1: Continuous (a) vs Discrete (b) signal representation.

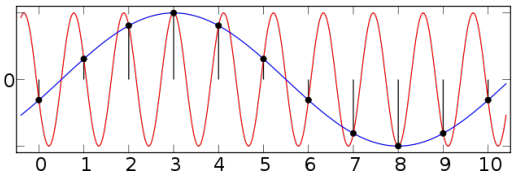


Figure 2.2: Example of aliasing due to low sample rate [37]

2.2.1 Discrete Fourier Transform

The Fourier Transform is a natural way to consider audio data as it provides a snapshot into what frequencies are present in a given audio sample. As digital audio data represents sound waves digitally, it makes sense to use a discrete form of the Fourier Transform.

The Discrete Fourier Transform (DFT) is a mathematical operation that projects discrete time-series data into the frequency domain. By doing this it finds a set of complex sinusoids whose corresponding frequency, amplitude and phase describe the original signal. As these form the base of building blocks for describing the original signal, they are often referred to as *basis functions*. Given a sequence of N samples $x_0, x_1, ..., x_{N-1}$, the DFT is defined by

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi kn/N} \tag{2.1}$$

$$= \sum_{n=0}^{N-1} x_n \cdot [cos(2\pi kn/N) - i \cdot sin(2\pi kn/N)] \tag{2.2}$$

where $k \in [0, N-1]$, and the expansion of $e^{-i2\pi kn/N}$ follows from Euler's formula. Each X_k is a complex value containing the magnitude and phase of the frequency $2\pi k/N$. By doing this the DFT decomposes a signal into a set of basis functions that represent the original signal. This representation allows one to easily see how much of certain frequencies are present in a given time domain audio sample. Figure 2.3 shows a plot of the intensity of each of the $N/2$ frequency bins resulting from the DFT of the signal in Figure 2.1b, using N = 512. The first thing apparent in Figure 2.3 is that there is not simply two spikes corresponding to the two frequencies present in the original signal and zeroes everywhere else. The reason for this is that the basis functions found to describe the signal do not contain functions that exactly correspond to the two frequencies present in the original signal. The implication of this is that frequencies which do not coincide with a base function will exhibit non-zero projections across the entire basis set [18]. This is known as *spectral leakage*, the effects of which, along with techniques for working within the leakage, are out of the scope of this project. For a more in-depth review of the effects of spectral leakage the reader is directed to Harris

[18].

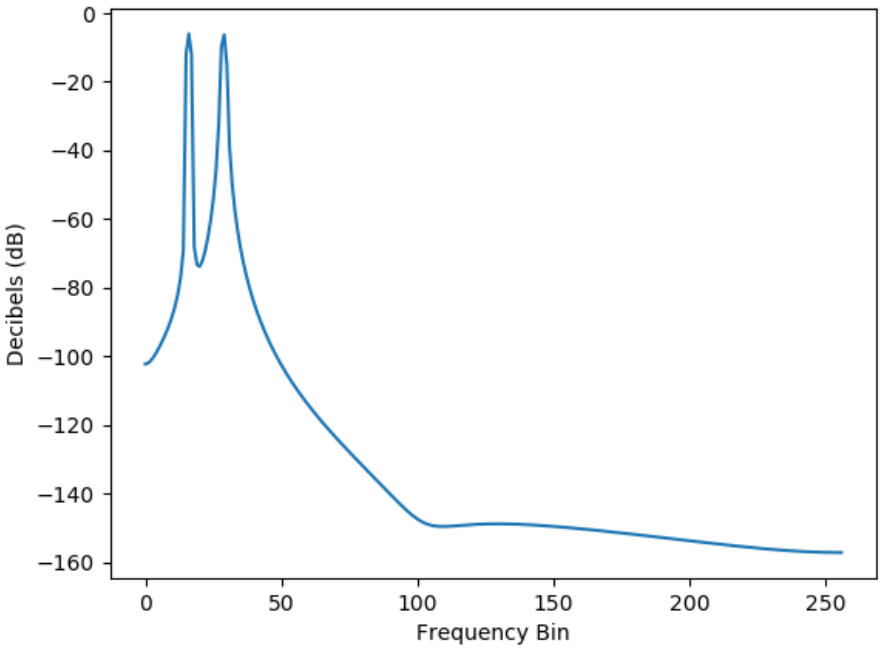


Figure 2.3: Discrete Fourier Transform (Magnitude) of 1kHz sine + 1.8kHz sine signal.

To provide complete understanding of the output in Figure 2.3, the concept of the decibel is presented. Simply put, the decibel is a way of expressing the ratio of one value to another. In the signal domain, it is a way of representing the strength of a signal component with respect to some reference value. It is convenient to take this value as the unit amplitude of a sinusoid, or 1 volt. All sinusoids present can now be measured by the ratio of their strength to the reference value. The decibel ratio can be calculated using the formula:

$$dB = 10\log_{10}(\frac{V_{reference}}{V_{signal}}). \tag{2.3}$$

Looking at this equation one can see that a decibel level of $-20dB$ corresponds to a power ratio of measured signal to reference level of $1 : 100$. This allows us to see that while spectral leakage results in a non-perfect basis function representation, the non-zero basis values are often negligible.

2.2.2 Short-Time Fourier Transform

The DFT gives us a view of the frequencies present in a snapshot in time of an audio signal. This is beneficial as it is easier to interpret and process than the raw audio signal, but does not take the time-varying nature of audio data into account. There

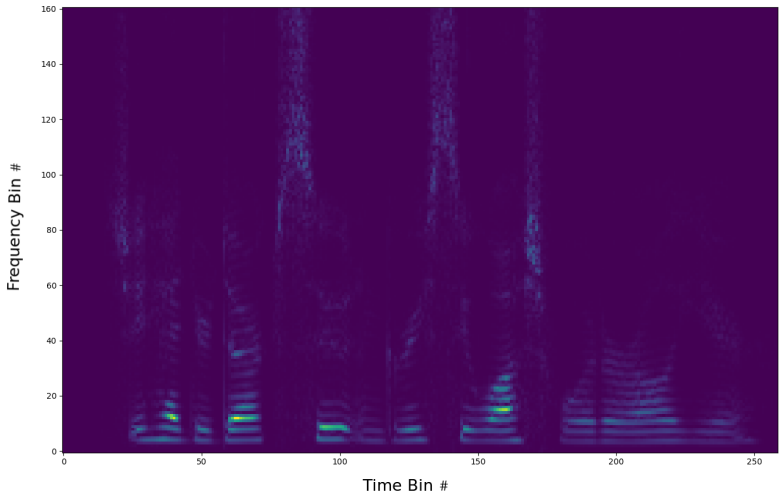


Figure 2.4: Magnitude spectrum of a TIMIT training utterance

may be frequencies early in an audio sample that are not present later on. In order to take this time-varying nature into account, the Short-Time Fourier Transform (STFT) is introduced.

To compute the STFT of an audio signal, the data is first divided up into windowed frames. The dividing up of the original signal is done by setting an analysis window size and time-stride distance[1], both measured in number of discrete samples. For each resulting window position, the DFT is computed for the samples present in that temporal slice of the audio signal. The resulting DFTs of each window are then combined to form the resultant STFT - a complex valued 2-D array containing the frequencies (magnitude) and phase (angle) of the sinusoids present in the audio signal and how they change over time. The parameter choices of the window size and time-stride have an effect on the resolution accuracy of the frequencies and temporal position of their attack and release. The wider the analysis window, the better the frequency resolution, as there are more samples being considered during any given DFT. This frequency resolution comes at a cost of less temporal accuracy, as the more samples considered means that the snapshot in time of a single DFT spans across a longer lapse in time. Choices of these parameters are application specific, and an in-depth discussion falls out of the scope of this project. Figure 2.4 shows the magnitude of the STFT for a randomly chosen TIMIT data set utterance.

Output from the STFT magnitude is 2-dimensional, relating an intensity value (similar to pixel intensity in a 1-channel image) to a particular frequency bin at a particular time. In this way, the STFT output can be treated, analysed, plotted and inspected much in the same way as traditional 1-channel images.

[1]The number of samples to move the window forward by before taking the next DFT.

2.2.3 Inverse Short Time Fourier Transform

We have seen that the STFT is able to output a spectrogram of complex values that encode both the frequency and phase of sine waves present in any given audio signal, as well as how they change over time. This result is useful for analysis of audio samples as it provides a glimpse into the frequencies present in a sample. It would be additionally useful to be able to make changes to this representation in the spectrogram domain, and see what effect this might have on the signal that the modified spectrogram represents. The Fourier inversion theorem shows that the Fourier transform is invertible provided the signal and its Fourier transform are integrable and continuous [12]. As such, the inverse STFT (iSTFT) exists and can be used to recover the original signal when provided with the complex spectrogram representation. In other words the magnitude and phase portions that make up the complex STFT output must be provided.

Griffin-Lim Signal Estimation Algorithm

In the case where only the magnitude portion of the STFT representation is known, an algorithm exists to recover an approximation of the missing phase portion and estimate a re-synthesised signal [16]. The algorithm is known as the Griffin-Lim algorithm and it builds a signal through an iterative process whereby the mean squared error between the re-built signal's STFT magnitude and the original STFT magnitude is minimised. The algorithm has been tested on speech data synthesis where it produced high-quality results [16].

2.3 Principal Component Analysis

Principal Component Analysis (PCA) is a linear mathematical technique that allows for the projection of a set of data points onto a lower dimensional yet orthogonal and decorrelated sub-space. By doing so, one can reduce the dimensionality of data points in a data set while still retaining an accurate and high measure of the variance within the original data set. With real data sets, it's often observed that not all variables have a great effect on the underlying information or phenomena of interest within the data [11]. In these cases, the information can be represented in some lower dimensional sub-space. Figure 2.5 illustrates the reduction of a 2-dimensional data set to a sub-space of 1-dimension. From this, one can intuitively see that the overall direction of the distribution of the data points explains the majority of the variance in the original data set, and represents the first principal component. More formally, given a set of data points $\{x_n\}$ where $n \in [1, \ldots, N]$, and x_n has dimensionality D, we wish to project the data onto a sub-space with dimensionality $M < D$.

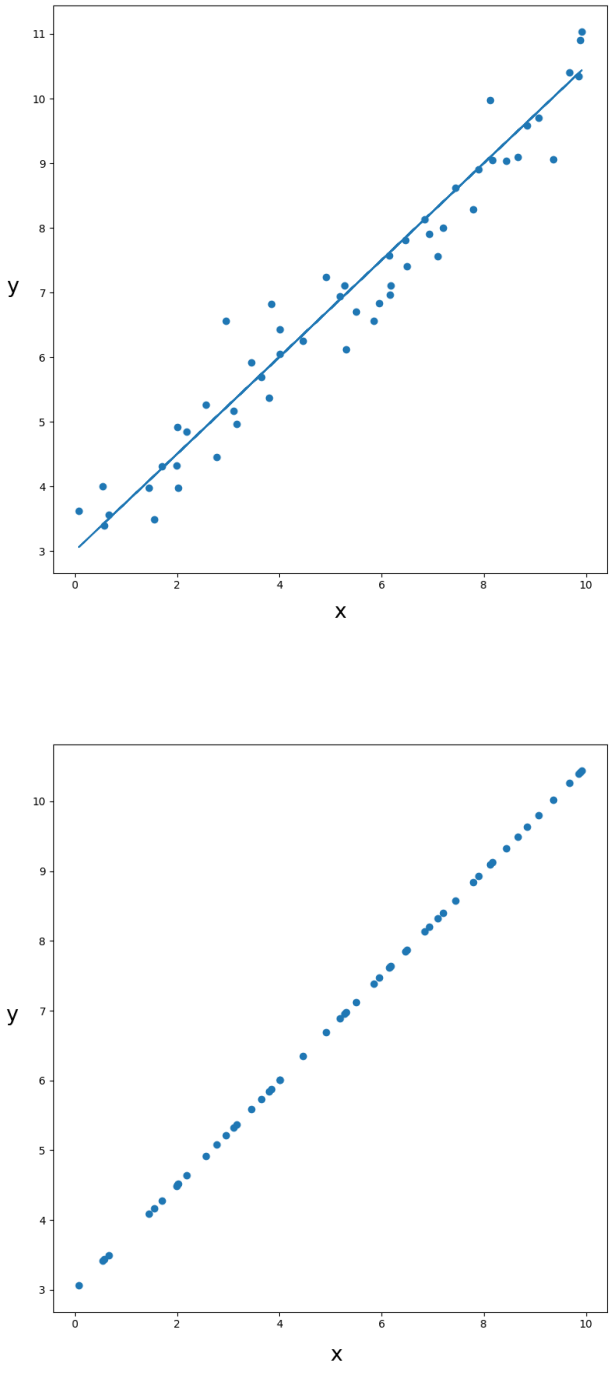


Figure 2.5: Reducing data dimensionality with PCA. Top: Scatter plot of random data with underlying linear function. Bottom: Data points are best described along $y = \frac{3}{4}x+3$. This direction can now be chosen as the new base direction and the problem is reduced from 2D to 1D.

Bishop [3] gives a mathematical formulation for computing principal components as follows. First, consider the projection where $M = 1$. The direction of this one-dimensional space can be defined by some D-dimensional vector u_1, which is constrained to be a unit vector such that $u_1^T u_1 = 1$. We project each original data point x_n onto a scalar value $u_1^T x_n$. We can calculate the mean of the projected data as $u_1^T \bar{x}$ where $\bar{x}$ is the mean of the original data set:

$$\bar{x} = \frac{1}{N} \sum_{n=1}^{N} x_n. \tag{2.4}$$

The variance of the projected data is therefore given by

$$\frac{1}{N} \sum_{n=1}^{N} (u_1^T x_n - u_1^T \bar{x})^2 = u_1^T S u_1, \tag{2.5}$$

where S is the data covariance matrix defined by

$$S = \frac{1}{N} \sum_{n=1}^{N} (x_n - \bar{x})(x_n - \bar{x})^T. \tag{2.6}$$

The goal now becomes finding the direction, u_1, that maximises the variance of the projected data. Taking the first derivative with respect to u_1 and setting it equal to zero to find a maximum, we get

$$S u_1 = \lambda_1 u_1. \tag{2.7}$$

From this, we can see that u_1 is an eigenvector of the covariance matrix S and that the initial constraint of $u_1^T u_1 = 1$ leads to

$$u_1^T S u_1 = \lambda_1, \tag{2.8}$$

which says that the variance will be maximised when we set u_1 equal to the eigenvector with the largest eigenvalue λ_1. This eigenvector is then known as the first principal component. Further principal components $u_2, \ldots, u_N$ where $N \leq D$ are then represented by the remaining eigenvectors of S. The magnitude of an eigenvector's eigenvalue indicates how much of the data set's variance occurs in the direction of the eigenvector. For dimensionality reduction, components with low respective eigenvalues can be discarded as those directions do not contribute significantly to the variance in the data [11]. In this way, we can map the original data points to the new coordinates given by the principal components used. We can also inversely recreate the data. There will obviously be some loss in the instances where fewer principal components are used than there are original dimensions. This loss is often more than acceptable given the benefits of reducing the original dimensionality.

2.4 Neural Networks

Artificial Neural Networks are an attempt to mathematically model the way information is encoded in the neural structure of the brain. Layers of artificial neurons are connected in linear combinations which are strengthened or weakened by their connection weighting and bias. These linear combinations are then passed through a non-linear "activation function" to arrive at the final output of a given neuron. Through supervised learning approaches[2], networks can learn to encode the manner in which some input variable $\boldsymbol{x}$ is transformed by some function f into the output variable $\boldsymbol{y} = f(\boldsymbol{x})$ [3]. By doing this, it is said that the neural network has learned the function f. This is useful as machine learning tasks involve the approximation of some function that maps a set of observed data inputs to a resultant data output. Once the function has been approximated, outputs can be predicted for newly observed input data, similar to other regression and classification tasks. The non-linearity of the activation function of each neuron is vital to the ability of the network to approximate non-linear functions. If the functions were linear, then the network would only be capable of approximating linear functions, as the repeated application of linear transformations is itself a linear transformation [3]. This can be thought of as a type of non-linear regression task.

2.4.1 Formalisation

In a generalised feed-forward network, each unit computes a weighted sum of its inputs of the form

$$a_j = \sum_i W_{ji} x_i + c_j \tag{2.9}$$

where x_i is an input neuron that has a connection to neuron j, and w_{ji} is the weight associated with that connection. More formally, each neuron in a neural network is a non-linear function of a weighted linear combination of its input values. Consider a neural network with an input layer of size D, output layer of size O, and one "hidden" layer[3] of size H connecting the two. Figure 2.6 shows a graphical representation of this network. We denote the input nodes to be $\boldsymbol{x}$, the activations and outputs at the hidden layer to be $\boldsymbol{a}$ and $\boldsymbol{h}$ respectively, and the activations and outputs at the output layer to be $\boldsymbol{b}$ and $\boldsymbol{y}$ respectively. The weights and biases connecting the input and hidden layer are denoted as $\boldsymbol{W}^{(1)}$ and $\boldsymbol{c}^{(1)}$ whereas the weights and biases connecting the hidden layer to the output layer are $\boldsymbol{W}^{(2)}$ and $\boldsymbol{c}^{(2)}$. We can thus define the activation value of

[2]Supervised learning involves knowing the target vector, or "answer", that we desire a network to learn to approximate when given a certain input.

[3]By convention, the layers between the input and output layers are called "hidden" layers, as they are the internal black-box of the neural network that map the input nodes to the output nodes by learning the underlying function f.

the j^{th} hidden unit to be

$$a_j = \sum_{i=1}^{D} W_{ji}^{(1)} x_i + c_j^{(1)}, j \in [1, H] \tag{2.10}$$

which written in matrix notation takes the form

$$\boldsymbol{a} = \boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{c}^{(1)} \tag{2.11}$$

where $\boldsymbol{W}^{(1)} \in \mathbb{R}^{H \times D}$ and $\boldsymbol{x} \in \mathbb{R}^{D}$. The output of the j^{th} hidden unit after passing through the unit's activation function is then

$$h_j = z_j^{(1)}(a_j) \tag{2.12}$$

or

$$\boldsymbol{h} = \boldsymbol{z}^{(1)}(\boldsymbol{a}) \tag{2.13}$$

where $z_j^{(1)}$ is the differentiable activation function chosen for hidden neuron j and $\boldsymbol{h} \in \mathbb{R}^{H}$. Similarly, for the output layer we get

$$b_k = \sum_{j=1}^{H} W_{kj}^{(2)} h_j + c_k^{(2)}, k \in [1, O] \tag{2.14}$$

or

$$\boldsymbol{b} = \boldsymbol{W}^{(2)}\boldsymbol{h} + \boldsymbol{c}^{(2)} \tag{2.15}$$

where $\boldsymbol{W}^{(2)} \in \mathbb{R}^{O \times H}$, and

$$y_k = z_k^{(2)}(b_k) \tag{2.16}$$

or

$$\boldsymbol{y} = \boldsymbol{z}^{(2)}(\boldsymbol{b}) \tag{2.17}$$

where $z_k^{(2)}$ is the activation function for output neuron k and $\boldsymbol{y} \in \mathbb{R}^{O}$.

It can now be seen that the function mapping $\boldsymbol{x}$ to $\boldsymbol{y}$ is parametrised by $\boldsymbol{W}^{(1)}$, $\boldsymbol{c}^{(1)}$, $\boldsymbol{W}^{(2)}$ and $\boldsymbol{c}^{(2)}$. We can define more generally the output of a neural network for a given input $\boldsymbol{x_n}$ to be the function defining a *composition* of functions that transform each layer:

$$\boldsymbol{y} = f(\boldsymbol{x_n}, \boldsymbol{W}, \boldsymbol{c}) = \boldsymbol{z}^{(2)}(\boldsymbol{W}^{(2)}\boldsymbol{z}^{(1)}(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{c}^{(1)}) + \boldsymbol{c}^{(2)}). \tag{2.18}$$

Given that the activation functions are chosen to be differentiable, it can be shown that the function defining the output of the neural network is differentiable and can be expanded via the application of the chain rule.

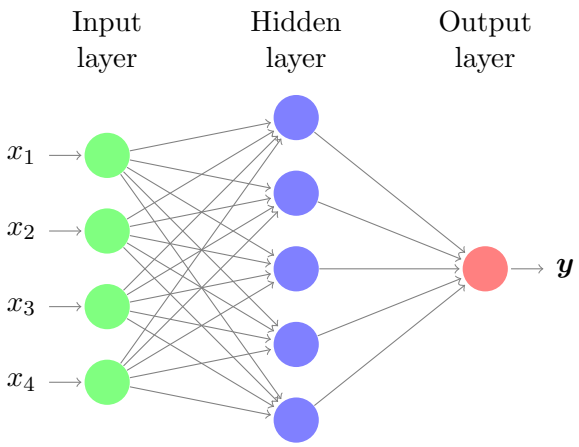


Figure 2.6: A typical NN topology with 1 hidden layer.

2.4.2 Activation Functions

Common choices for the activation function are the logistic sigmoid:

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{2.19}$$

or the hyperbolic tangent $\tanh(x)$ as these force the output to the ranges $[0, 1]$ and $[-1, 1]$ respectively, thereby normalizing the output at each layer, preventing values from growing too large and dominating the network. The $\tanh(x)$ activation function, in particular, has been shown to speed up convergence in the popular Back Propagation (BP) training algorithm [19]. Both functions are also popular choices thanks to their convenient derivatives which take the form

$$\frac{d}{dx}\sigma(x) = \sigma(x)(1 - \sigma(x)) \tag{2.20}$$

and

$$\frac{d}{dx}\tanh(x) = 1 - \tanh^2(x) \tag{2.21}$$

respectively. Looking at these derivatives one can see that they can be written in terms of the original function, thus allowing a virtually computation-free gradient value to be calculated. Another common choice, especially for the final output of the network, is the softmax function. Softmax behaves much like the logistic sigmoid function, squeezing outputs between $[0, 1]$, but enforces that all the output nodes sum to 1. This allows one to treat the output activations as a probability distribution over the predicted class labels.

2.4.3 Training

Given a set of data vectors $\{x_n\}$ where $n = 1, \ldots, N$ and corresponding set of target vectors $\{t_n\}$, training the network to approximate a function that maps the input training samples to the target vectors involves minimising a relevant *error* or *loss* function that penalises the difference between the output value of the network $\boldsymbol{y} = f(\boldsymbol{x_n}, \boldsymbol{W}, \boldsymbol{c})$ and the ground truth of the target vector t_n $\forall n \in [1, \ldots, N]$. One such error function is

$$E(\boldsymbol{W}, \boldsymbol{c}) = \frac{1}{2} \sum_{n=1}^{N} \|\boldsymbol{y} - t_n\|^2. \tag{2.22}$$

Minimizing this error function learns the set of weight and bias parameters that encode an approximated best fit to the underlying function in the training data.

A common approach to training a neural network given the training data is to use the Gradient Descent (GD) algorithm to minimise the chosen error function. Gradient Descent is a well-known minimisation technique established in the late 1800s to early 1900s [7, 17]. It assumes that the gradient of the function being minimised is calculable and tractable at all training samples, and that the parameters will be updated by a very small step towards a value that would minimise the function. The gradient of this step is calculated from all the data points simultaneously. The update rules for the weights and biases are intuitive and can be formalised as follows:

$$\boldsymbol{W} \Leftarrow \boldsymbol{W} - \alpha \cdot \frac{\partial E(\boldsymbol{W}, \boldsymbol{c})}{\partial \boldsymbol{W}} \tag{2.23}$$

$$\boldsymbol{c} \Leftarrow \boldsymbol{c} - \alpha \cdot \frac{\partial E(\boldsymbol{W}, \boldsymbol{c})}{\partial \boldsymbol{c}} \tag{2.24}$$

where α is known as the *learning rate*, and is chosen to be a small number such that $\alpha \ll 1$. This ensures very small updates are made to the parameters, and promotes convergence by preventing the updates from over-stepping[4]. As the error function approaches a minimum, its gradient approaches zero and the parameters converge on values that satisfy the minimisation of the error function.

One common issue with traditional GD is that calculating the gradient from all the available data points simultaneously is often intractable. To overcome this, Stochastic Gradient Descent (SGD) and Batch Gradient Descent (Batch-GD) can often be substituted. Stochastic Gradient Descent works by calculating the gradient of the loss function with respect to a single randomly selected training sample at a time. This allows for computational feasibility, but can take a longer time to converge as the gradient with respect to the single training sample may be very different from the true

[4]Over-stepping is a term used to describe the situation in which the learning rate is too large, resulting in each update being too large to move toward the true local minimum, and over-stepping it. This can even result in a chaotic system that diverges away from the local minimum.

gradient across the entire data set. To smooth out such gradients, Batch-GD calculates the average gradient across a random batch of the data points. Both SGD and Batch-GD have been shown to successfully converge to a local minimum given a low enough learning rate [4].

2.4.4 Back Propagation

Given that the training of a neural network is underpinned by finding the derivative of an error function with respect to a parameter, which in turn involves finding the derivative of the network output with respect to that parameter, the next practical step we need is a method with which to efficiently calculate that gradient. This is what a technique known as Back Propagation (BP) has been designed to do. Back Propagation allows for the calculation of the amount of error that each neuron adds to the overall network error by propagating the overall output error backward through the network, which once again involves simple linear algebraic operations.

The computed activation a_j of output unit j is transformed by a non-linear activation function $z(\cdot)$ to give the output h_j of the form

$$h_j = z(a_j). \tag{2.25}$$

We now assume that the activations at all units have been calculated during the forward-propagation step of pushing an input sample through the network to arrive at the output for the given state of the network.

Consider the derivative of E_n (the error for input sample n) with respect to a single weight w_{ji}. Application of the chain rule for partial derivatives yields

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} \tag{2.26}$$

Useful notation is defined:

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} \tag{2.27}$$

Using (2.9) we can write

$$\frac{\partial a_j}{\partial w_{ji}} = x_i \tag{2.28}$$

Substituting (2.27) and (2.28) into (2.26), we then get

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j x_i. \tag{2.29}$$

Intuitively this shows us that the required derivative is obtained simply by multiplying the value of δ for the unit at the output end of the weight by the value of x for the unit at the input end of the weight. We can, therefore, see that in order to calculate the

gradient we simply need to calculate the value of δ for each hidden and output units in the network, and then apply (2.29). Note that this implies that all the activations at every layer need to be stored during the forward pass. Therefore, while this approach leads to a computationally efficient algorithm for computing the gradient, it comes at a higher memory cost of having to keep track of previously computed values.

For the output units, we can see from our error function defined in Equation 2.22 that

$$\delta_k = y_k - t_k. \tag{2.30}$$

For the other hidden units, we once again start at the application of the chain rule for partial derivatives

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} \tag{2.31}$$

where the sum is over all k units to which unit j sends a connection. Substituting the definition of σ in (2.27), and making use of (2.9) and (2.25), we obtain a generalised formula for BP:

$$\delta_j = z'(a_j) \sum_k w_{kj} \delta_k. \tag{2.32}$$

In this manner, Bishop [3] outlines an algorithm as follows:

Algorithm 1 BackPropagation algorithm

1: Apply an input vector x_n to the network and forward propagate through the network using (2.9) and (2.25) to find the activations of all the hidden and output units.
2: Evaluate the δ_k for all output units using (2.30)
3: Backpropagate the δ's using (2.32) to obtain δ_j for each hidden unit
4: Use (2.29) to evaluate the required derivatives.

2.4.5 Dimensionality Reduction

Neural Networks can be thought of as dimensionality reducers or feature extractors [23]. If each consecutive hidden layer maps to a lower dimensionality and the network is successfully trained, then a more compact representation of the input data is encoded into the network at any given layer. This is seen in Gatys et al. [15] where the output at deeper and deeper layers represent the content of the input image, albeit in less detail. Hinton and Salakhutdinov [23] illustrates that auto-encoders, a type of NN architecture which can be trained to map down the dimensionality of the input and from this re-construct the original input, are very effective for dimensionality reduction. Restricted Boltzmann Machines, while similar to auto-encoders in their generative ability to recreate their inputs, are trained greedily layer-by-layer rather than with full back-propagation of gradients [1]. As the training of an RBM occurs layer

by layer, the back-propagation step at any layer involves a much simpler chain-rule expansion. Higher layers also receive outputs from fully trained lower layers when it comes to training them, which is useful when learning a hierarchical representation [32]. If a neural network is able to encode information in its input in such a way that the representations it creates at its output layer can be backpropagated through the network to create an accurate reconstruction, then the output layer representations can be seen as features extracted from the data. In order to provide a description of RBMs, a few preliminary concepts must be introduced.

2.5 Bayesian Probability Model

An important concept in the core derivations of inference in Restricted Boltzmann Machines, and more generally in the discipline of pattern recognition as a whole[5], is that of probability theory and ultimately Bayes' Theorem, which will be discussed in this section.

2.5.1 Sum and Product Rules of Probabilities

Consider two bags, one red and one blue, both containing different numbers of both red and blue balls. We perform N random ball selections, mark down the colour of the bag and the ball, and replace the ball into the bag from which it came. Any ball in a given bag is as likely as being picked as any other in that bag. Let the number of times a ball of colour x was chosen from a bag of colour y be denoted $n_{x,y}$ where $x, y \in \{red, blue\}$. Clearly, the probability of a random choice resulting in a ball of colour x being chosen from a bag of colour y is

$$P(ball, bag) = P(x, y) = \frac{n_{x,y}}{N}. \tag{2.33}$$

Similarly, the probability of a ball of colour x being chosen is

$$P(ball) = P(x) = \frac{c_x}{N} \tag{2.34}$$

where c_x is the number of times a ball of colour x was picked, regardless of the colour of the bag it came out of. Intuitively we can make the connection that as c_x is the number of times a ball of colour x was chosen, regardless of the colour of the bag y, that $c_x = \sum_y n_{x,y}$. In this way, we can define the *sum rule* of probabilities as follows: Given two random variables X and Y, the probability of X taking on the value x, regardless of the value of the variable Y is given by

$$P(x) = \sum_y P(x, y). \tag{2.35}$$

This is known as *marginalisation*, and it is said that we have marginalised out the variable y from the the probability distribution $P(x, y)$.

If we consider only the instances where the chosen ball was of colour x, and wish to know the fraction of these instances where the bag it was chosen from was of colour y, then we are looking for the *conditional* probability of the bag colour given that the

[5]Pattern recognition involves a fair amount of *uncertainty*, be it from noisy measurements or small data sets that do not cover the breadth of the problem domain, and requires a probabilistically sound way of working within this uncertainty [3].

ball was of colour x. This is written as

$$P(Bag = y|Ball = x) = \frac{n_{x,y}}{c_x}. \tag{2.36}$$

From (2.33), (2.34) and (2.36) we can define the *product rule* of probabilities as follows. Given two random variables X and Y, the joint probability of X taking on the value x *and* Y taking on the value y is given by

$$P(x, y) = \frac{n_{x,y}}{N} = \frac{n_{x,y}}{c_x} \cdot \frac{c_x}{N} \tag{2.37}$$

$$= P(y|x) \cdot P(x) \tag{2.38}$$

2.5.2 Bayes' Theorem

Bayes' theorem involves relating conditional probabilities of random variables in such a way that prior known probability information can be used to get a more accurate measure of an unknown probability. The equation follows simply from the product rule (2.38), and can be arrived at as follows:

$$P(x, y) = P(y|x) \cdot P(x) \tag{2.39}$$

and

$$P(x, y) = P(x|y) \cdot P(y) \tag{2.40}$$

then

$$P(y|x) \cdot P(x) = P(x|y) \cdot P(y) \tag{2.41}$$

$$P(y|x) = \frac{P(x|y) \cdot P(y)}{P(x)}. \tag{2.42}$$

2.6 Probabilistic Graphical Models

Probabilistic Graphical Models (PGMs), often just referred to as Graphical Models (GMs), are a way of representing probability distributions with graphs consisting of nodes and edges. Each random variable in the probability distribution is represented by a node and dependencies (or independences) between variables are represented by edges connecting one node to another (or lack thereof). There are clearly many different structures a graphical model can take, but certain structures give rise to classes of graphical models with different properties and procedures for statistical inference[6]. If a joint probability distribution can be factorised in such a way that its factorisation can be represented by a class of graphical model, then it is said that the graphical model

[6]Inference is an aptly named concept that refers to the ability to *infer* values of certain random variables given others.

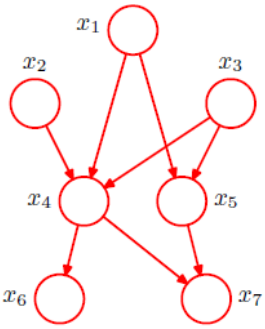


Figure 2.7: Example DAG describing joint probability over variables x_1 to x_7 [3]

describes that probability distribution. This is attractive as we can gain intuition about a given problem by inspecting the graphical model that describes it, and we can use well-established graph theory concepts to develop effective algorithms for complex computations on the model. Two common graphical models are Bayesian networks and Markov Random Fields.

2.6.1 Bayesian Networks

Bayesian networks are specifically Directed Acyclic Graphs (DAGs) representing causal relationships between variables by directed edges from one to the other. Consider the following joint probability distribution factorisation:

$$p(\boldsymbol{x}) = p(x_1)p(x_2)p(x_3)p(x_4|x_1, x_2, x_3)p(x_5|x_1, x_3)p(x_6|x_4)p(x_7|x_4, x_5). \tag{2.43}$$

This probability distribution can be described by the graphical model in Figure (2.7), which is a DAG, and therefore a Bayesian Network, in which values can be inferred for any variable, given that we know the values of the variables represented by its parent nodes.

The joint distribution defined by a graph is computed as the product of conditional distributions of each node in the graph, conditioned on the values of its parent nodes. [3]. Therefore, for a DAG we can more generally represent the joint probability distribution of a graph consisting of K nodes as follows:

$$p(\boldsymbol{x}) = \prod_{k=1}^{K} p(x_k|\text{parents}(x_k)) \tag{2.44}$$

where parents(x_k) represents the set of parents of x_k, and $x = \{x_1, \ldots, x_K\}$.

2.6.2 Markov Random Fields

Markov Random Fields (MRFs) are another type of graphical model that use undirected graphs, which may or may not contain cycles, to describe sets of random variables in a probability distribution. In order for a set of random variables to be considered an MRF there are a few properties, known as Markov properties, that must hold. Conversely, if a set of random variables satisfies the Markov properties, then it can be considered to be an MRF. Knowing that a given probability distribution satisfies one or more of the Markov properties allows us to make assumptions about conditional independence between variables or groups of variables in the distribution. The property definitions that follow are all made considering a graph $G(V, E)$ used to describe a set of random variables $\boldsymbol{X} = (X_v)$ where X_v is the random variable associated with node $v \in V$.

Pairwise Markov Property

If the pairwise Markov property holds, the random variables represented by non-adjacent nodes are conditionally independent given all other nodes:

$$X_a \perp\!\!\!\perp X_b \mid X_{V\setminus\{a,b\}} \quad \forall\{a, b\} \notin E \tag{2.45}$$

Local Markov Property

If the local Markov property holds, then for all $v \in V$, the associated random variable X_v is conditionally independent of all other variables given its neighbourhood[7] X_w where $w \in N_v$ [9]:

$$X_v \perp\!\!\!\perp X_{V\setminus\{N_v\}} \mid X_{N_v} \quad \forall v \in V. \tag{2.46}$$

Global Markov Property

Suppose we identify three *disjoint* sets of nodes $A, B, C \subset V$ such that C separates A and B [8]. The (global) Markov property holds for the distribution if, for all such sets in the graph, the variables $(X_a)_{a\in A}$ are conditionally independent of the variables $(X_b)_{b\in B}$ given $(X_c)_{c\in C}$:

$$(X_a)_{a\in A} \perp\!\!\!\perp (X_b)_{b\in B} \mid (X_c)_{c\in C} \tag{2.47}$$

if C separates A and B.

The above properties allow insight into conditional independences of random variables in a probability distribution if they can be met. The properties, however, are non-trivial to establish for an arbitrary probability distribution. As outlined in Bishop [3] and Fischer and Igel [9], a commonly used process called clique factorisation can be

[7] The neighbourhood of v, denoted N_v is the set of all vertices directly adjacent to v.

[8] The set of nodes C separates A and B if there exists no path from any node in set A to any node in set B that does not pass through a node in set C.

employed to decompose a distribution described by an MRF graphical model G into products of potential functions over the maximal cliques[9] of the graph. In other words, for an MRF, we can describe the joint distribution as follows:

$$p(\boldsymbol{x}) = \frac{1}{Z} \prod_{C} \Psi_C(x_C) \tag{2.48}$$

where C is from the set of maximal cliques of the graph. Here, Z is a normalisation constant often referred to as the *partition function* and is defined by

$$Z = \sum_{x} \prod_{C} \Psi_C(x_C) \tag{2.49}$$

and ensures that $p(\boldsymbol{x})$ is correctly normalised. Learning a model now involves learning the parameters of the potential functions such that observed configurations of $\boldsymbol{x}$ yield a high probability. As the potential functions are arbitrary, we restrict the learned potential functions to be strictly positive by enforcing $\Psi_C(x_C) = e^{-E(x_C)}$ where E is an arbitrary energy function chosen in such a way as to assign low energy to observed configurations of variables. This lets us arrive at the energy based representation of the joint probability distribution of

$$p(\boldsymbol{x}) = \frac{1}{Z} e^{-E(\boldsymbol{x})}. \tag{2.50}$$

Here we see a relationship established whereby if some sample $\boldsymbol{x}$ yields a high probability $p(\boldsymbol{x})$, then its corresponding energy value $E(\boldsymbol{x})$ will be low, and vice versa.

Example Application

Bishop [3] discusses an application of a Markov random field to the problem of image de-noising. The problem sets up a MRF where each binary pixel in a noisy image represents one random variable. It is supposed that the binary pixels in an unknown noise-free image each represent another random variable. The corresponding pixels (variables) in each image are connected to define a dependence that the pixels in the noise-free image may swap their binary value with some small probability (transition probability). Given the noisy image, the goal is to reconstruct the noise-free image. Through clever construction of a heuristic energy function for the network that ensures less noisy images have lower energy, a sample for the noise-free image can be generated.

[9] In graph theory, a clique is a group of fully connected nodes in a graph. A maximal clique is a clique such that the addition to the set of any other node in the graph renders the resulting set no longer a clique.

2.6.3 Generative Models

Generative Models are models that approximate the *joint* distribution of inputs *and* outputs. By doing this, it is possible to generate synthetic data points in the input space by sampling from it [3]. In other words, generative models learn to model the *joint* probability distribution across the inputs and outputs, $p(\text{inputs}, \text{outputs})$ as opposed to discriminative methods which model the conditional distribution $p(\text{inputs}|\text{outputs})$.

An interesting generative model for audio data is presented in Oord et al. [40]. The generated samples from the trained network are incredibly high quality time-domain audio signals. The inputs to the model cover not only the audio signal of a given training sample, but also the textual transcription of the speech content along with an ID of the person speaking. When generating new signals without providing a textual transcription to condition the model on during the sampling process, the authors note that the generated signal "results in a kind of babbling, where real words are interspersed with made-up word-like sounds" [40]. When the sampling process includes a textual transcription on which to condition the network, the generated audio signal contains speech content matching the textual transcription. Similarly, the authors can change the voice with which the speech is generated by conditioning the network on a given speaker ID from the training set.

2.6.4 Drawing Samples

We have now discussed Graphical Models as a way to describe factorisations of a probability distribution which allows us to make certain assumptions about the conditional independence between random variables in the distribution. We would now like to generate samples from the underlying distribution described by the models. In other words, we would like to sample some configuration of $\boldsymbol{x}$ such that $p(\boldsymbol{x})$ is high. Doing so turns out to be intractable, largely due to the normalisation constant Z, which sums over all possible configurations of the random variables. As always, when faced with a problem in which exact measurements are intractable, we turn to methods that allow us to approximate the result.

Markov Chain Monte Carlo Methods

Markov chains are stochastic processes that involve random variables that can take on certain states. Each variable state has a set of *transition probabilities* defining how likely its state will transition from the current value to another value. The important restriction in Markov chains is that the transition probabilities of the variables at state S_{n+1} are dependent only on the values of the variables at state S_n. In other words, the transition probabilities are dependent only on the current state of the chain, regardless of any of the states that have come before. In order to set up a given Markov

chain, one needs to define the probability distribution for the initial state, together with the conditional probabilities for subsequent variables in the form of transition probabilities [3]. As the chain becomes longer and longer, assuming the chain satisfies certain properties, the probability of a given configuration of variables converges to an equilibrium.

Markov Chain Monte Carlo (MCMC) methods are sampling approaches that centre around setting up a Markov chain starting at a given state, S_0, and calculating the next state in the chain a certain number of times, for instance arriving at S_N. As outlined in Fischer and Igel [9], if the chain satisfies the *irreducible* and *aperiodic* conditions of Markov chains[10], as the number of steps in the chain approaches infinity, the difference between the sampled probability density and the actual underlying probability density becomes zero [5]. In this way, we are able to draw estimated samples from the complex underlying probability distribution defined by the Markov process

Gibbs Sampling

One widely used MCMC sampling algorithm relevant to the problem at hand is Gibbs sampling. Consider a joint probability distribution that satisfies Markov properties and is written as $p(\boldsymbol{x}) = p(x_1, x_2, \ldots, x_N)$. Suppose we have constructed a Markov chain with some chosen initial state of the variables $\boldsymbol{x}$. We now repeatedly replace the value of one of the random variables by sampling the new value from the distribution of that variable conditioned on all the other variables. In other words, we sample $p(x_i|\boldsymbol{x}\backslash\{x_i\})$. After each step we change which variable we are sampling for the next step. Bishop [3] shows that, after sufficiently many iterations, this procedure successfully samples from the required distribution.

2.7 Restricted Boltzmann Machines

Restricted Boltzmann Machines are a type of generative Markov network which can conveniently be thought of as a neural network. Generative models are able to *generate* samples from the joint distribution of its variables, for example $p(v, h)$, as opposed to discriminative models which calculate *conditional* probabilities, for example $p(h|v)$. The network consists of two sets of random variables $\boldsymbol{v}$ and $\boldsymbol{h}$. $\boldsymbol{v}$ represents the observable data in a data set, and $\boldsymbol{h}$ is a set of latent, or hidden, variables used to capture the dependencies between the observed variables. Each $v_i \in \boldsymbol{v}$ is fully connected to each $h_j \in \boldsymbol{h}$ and vice versa. The simplest RBM deals with input and output variables that are binary-valued random variables. In other words, $v_i, h_j \in \{0, 1\}$.

[10] An extended discussion of Markov chains is beyond the scope of this dissertation, for more detail see Fischer and Igel [9] and Bishop [3].

For this binary-binary RBM, each joint configuration of input $\boldsymbol{v}$ and output $\boldsymbol{h}$ is assigned an energy governed by the energy function:

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\sum_{i,j} v_i W_{ij} h_j - \sum_j bjhj - \sum_i c_i v_i \tag{2.51}$$

$$= -\boldsymbol{h}^T W \boldsymbol{v} - \boldsymbol{b}^T \boldsymbol{h} - \boldsymbol{c}^T \boldsymbol{v} \tag{2.52}$$

where W_{ij} is the weight connecting v_i to h_j, b_j are hidden unit biases and c_i are visible unit biases. The probability of a joint combination of input $\boldsymbol{v}$ and output $\boldsymbol{h}$, as per (2.50), is defined to be:

$$p(\boldsymbol{v}, \boldsymbol{h}) = \frac{1}{Z} e^{-E(\boldsymbol{v}, \boldsymbol{h})} \tag{2.53}$$

where Z is a normalisation function, summing over all possible combinations of $(\boldsymbol{v}, \boldsymbol{h})$ pairs. This introduces the relationship that vectors $\boldsymbol{v}$ belonging to the set of data modelled by the network, i.e. $p(\boldsymbol{v}, \boldsymbol{h})$ is high, should have a low corresponding energy. Finding the right weight matrix $\boldsymbol{W}$ and biases $\boldsymbol{c}$ and $\boldsymbol{b}$ that enforces this relationship for the training data is how the model is trained.

2.7.1 Inference Derivations

From the definition of the energy function and the joint probability distribution, Larochelle [27] derives the conditional probabilities $p(h_j = 1|\boldsymbol{v})$ and $p(v_i = 1|\boldsymbol{h})$ as follows:

$$p(\boldsymbol{h}|\boldsymbol{v}) = \frac{p(\boldsymbol{v},\boldsymbol{h})}{p(\boldsymbol{v})} \tag{2.54}$$

$$= \frac{p(\boldsymbol{v},\boldsymbol{h})}{\sum_{h'} p(\boldsymbol{v},h')} \tag{2.55}$$

$$= \frac{\exp(\boldsymbol{h}^T W\boldsymbol{v} + \cancel{c^T\boldsymbol{v}} + b^T\boldsymbol{h})/\cancel{Z}}{\sum_{h'\in\{0,1\}^H} \exp(h'^T W\boldsymbol{v} + \cancel{c^T\boldsymbol{v}} + b^T h')/\cancel{Z}} \tag{2.56}$$

$$= \frac{\exp(\sum_j h_j W_j\boldsymbol{v} + b_j h_j)}{\sum_{h'_1\in\{0,1\}} \cdots \sum_{h'_H\in\{0,1\}} \exp(\sum_j h'_j W_j\boldsymbol{v} + b_j h'_j)} \tag{2.57}$$

$$= \frac{\prod_j \exp(h_j W_j\boldsymbol{v} + b_j h_j)}{\sum_{h'_1\in\{0,1\}} \cdots \sum_{h'_H\in\{0,1\}} \prod_j \exp(h'_j W_j\boldsymbol{v} + b_j h'_j)} \tag{2.58}$$

$$= \frac{\prod_j \exp(h_j W_j\boldsymbol{v} + b_j h_j)}{(\sum_{h'_1\in\{0,1\}} \exp(h'_1 W_1\boldsymbol{v} + b_1 h'_1)) \ldots (\sum_{h'_H\in\{0,1\}} \exp(h'_H W_H\boldsymbol{v} + b_H h'_H))} \tag{2.59}$$

$$= \frac{\prod_j \exp(h_j W_j\boldsymbol{v} + b_j h_j)}{\prod_j (\sum_{h'_j\in\{0,1\}} \exp(h'_j W_j\boldsymbol{v} + b_j h'_j))} \tag{2.60}$$

$$= \prod_j \frac{\exp(h_j W_j\boldsymbol{v} + b_j h_j)}{1 + \exp(W_j\boldsymbol{v} + b_j)} \tag{2.61}$$

$$= \prod_j p(h_j|\boldsymbol{v}) \tag{2.62}$$

where we get (2.54) from Bayes' theorem (2.42), (2.55) from the definition of marginal probability and (2.56) follows from (2.52). We arrive at (2.58) as the sum of exponents is the product of bases, and the simplification to (2.61) is due to h'_j taking on one of two values, 0 or 1. We can similarly show that

$$p(\boldsymbol{v}|\boldsymbol{h}) = \prod_i p(v_i|\boldsymbol{h}). \tag{2.63}$$

We now continue the derivation of $p(h_j = 1|\boldsymbol{v})$ as follows:

$$p(h_j|\boldsymbol{v}) = \frac{\exp(h_j W_j\boldsymbol{v} + b_j h_j)}{1 + \exp(W_j\boldsymbol{v} + b_j)} \tag{2.64}$$

$$p(h_j = 1|\boldsymbol{v}) = \frac{\exp(W_j\boldsymbol{v} + b_j)}{1 + \exp(W_j\boldsymbol{v} + b_j)} \tag{2.65}$$

$$= \frac{\exp(W_j\boldsymbol{v} + b_j)}{1 + \exp(W_j\boldsymbol{v} + b_j)} \times \frac{\exp(-W_j\boldsymbol{v} - b_j)}{\exp(-W_j\boldsymbol{v} - b_j)} \tag{2.66}$$

$$= \frac{1}{1 + \exp(-W_j\boldsymbol{v} - b_j)} \tag{2.67}$$

$$= \text{sigm}(W_j\boldsymbol{v} + b_j) \tag{2.68}$$

and similarly we can show

$$p(v_i = 1|\boldsymbol{h}) = \text{sigm}(W_i\boldsymbol{h} + c_i). \tag{2.69}$$

exp Now that we can infer the probability of a hidden unit being 1 given the visible units, and a visible unit being on given the hidden units, we can get an actual sample of each unit by sampling from the binomial distribution with the expected probability of getting a 1.

2.7.2 Training

Training an RBM to encode a particular distribution or data set now involves finding values for the parameters $\boldsymbol{W}$, $\boldsymbol{b}$ and $\boldsymbol{c}$ such that a known sample from the distribution, $\boldsymbol{v}$, will result in a high probability, and therefore a low energy, when it is presented to the model as input. The measure above that we wish to maximise is known as the *likelihood* of the model under the given data $\boldsymbol{v}$ and is simply written as $p(\boldsymbol{v})$. More often than not, training algorithms deal semantically with *minimisation* instead of *maximisation* of a function. Instead of maximising the likelihood $p(\boldsymbol{v})$, we speak of minimising the *negative* likelihood $-p(\boldsymbol{v})$. Finally, for mathematical convenience, we decide to minimise the natural logarithm of the negative likelihood. This is possible as the logarithm is a monotonic function, meaning that it achieves its maximum value at the same points as the function it operates on. The logarithm also makes the likelihood easier to handle as multiplications become sums, and small values are converted to larger negative values, which computers are able to handle with better precision. The model is trained via GD using the negative log-likelihood, $-logp(\boldsymbol{v})$, as the objective function to minimise. Through derivation and applying GD we get (for some parameter θ):

$$\frac{\partial - logp(\boldsymbol{v})}{\partial\theta} = E_h[\frac{\partial E(\boldsymbol{v}^{(t)}, \boldsymbol{h})}{\partial\theta}|\boldsymbol{v}^{(t)}] - E_{\boldsymbol{v},\boldsymbol{h}}[\frac{\partial E(\boldsymbol{v}, \boldsymbol{h})}{\partial\theta}] \tag{2.70}$$

where $E_{\boldsymbol{h}}$, the 'positive' phase, denotes the expectation under the data sample, and $E_{\boldsymbol{v},\boldsymbol{h}}$, the 'negative phase', denotes expectation under the current model. In all but the most trivial case, the negative phase is intractable to compute.

In order to utilise this gradient update feasibly, an estimate must be made for the negative phase. Hinton [22] introduces Contrastive Divergence (CD), an algorithm for estimating the negative phase. As an RBM can be seen as a Markov random field, Markov Chain Monte Carlo methods may be utilised in order to sample from its underlying distribution. CD involves clamping the RBM visible units with a sample from the training set $\boldsymbol{v}^{(t)}$, and inferring the value of the hidden units h. With this inferred value of the hidden units h we initialise a Markov chain and run a cycle of Gibbs-sampling to infer $\tilde{\boldsymbol{v}}$ and $\tilde{\boldsymbol{h}}$. This involves inferring the visible units $\tilde{\boldsymbol{v}}$ given $\boldsymbol{h}$,

and then inferring the hidden units $\tilde{\boldsymbol{h}}$ once more given $\tilde{\boldsymbol{v}}$. $\boldsymbol{v}^{(t)}$ and $\boldsymbol{h}$ now form the positive-phase samples, and $\tilde{\boldsymbol{v}}$ and $\tilde{\boldsymbol{h}}$ form the negative-phase samples.

Following this derivation with $\theta = W_{ij}$, c_i and b_j we get the following parameter update rules:

$$W \Leftarrow W + \alpha(\boldsymbol{h}\boldsymbol{v}^{(t)^T} - \tilde{\boldsymbol{h}}\tilde{\boldsymbol{v}}^T) \tag{2.71}$$

$$b \Leftarrow b + \alpha(\boldsymbol{h} - \tilde{\boldsymbol{h}}) \tag{2.72}$$

$$c \Leftarrow c + \alpha(\boldsymbol{v}^{(t)} - \tilde{\boldsymbol{v}}) \tag{2.73}$$

where α is some small learning rate, and $\alpha \ll 1$.

2.7.3 Sampling Procedures

n-Step Contrastive Divergence

As discussed in the previous section, training an RBM involves minimising the difference between the network expectation when conditioned on a training sample (positive phase), and the network expectation when conditioned on an actual sample from the model (negative phase). The above description of Contrastive Divergence, or CD-1, is a specific case of a more general sampling approach which we can denote CD-n, where n represents the number of steps of Gibbs sampling performed in order to arrive at a final estimate of the sample. As $n \to \infty$ the estimated sample approximates the model's true underlying distribution, as discussed in Section 2.6.4. It is not feasible, however, to run an infinitely long, or more realistically a very large ($n \gg 1$), chain of Gibbs sampling steps for every weight update. While it has been noted that a single-step of Gibbs sampling (CD-1) is, in general, sufficient for successful training of an RBM [1, 22], it is also clear that this approach results in rather poor estimations of the model's underlying distribution [21]. While the generative nature of the model may suffer as the true distribution has not been learned, this approach can lead to useful features being learned for classification.

Persistent Contrastive Divergence

As a result of the limitations of CD-1 to accurately estimate samples from the model's true distribution during training time, other sampling procedures have been developed. Persistent Contrastive Divergence (PCD) involves setting up a persistent Markov chain, named "persistent" as we do not re-initialise the chain with every new training sample. When calculating a weight update, a sample is drawn from the Markov chain by running 1 step of Gibbs sampling. This sample is then used as the sample in the negative phase. When the next sample is requested, another step of Gibbs sampling is performed starting from the state that the chain was left in after drawing the previous sample.

The intuition behind this approach is that between parameter updates, the model only changes very slightly, and therefore initialising the Markov chain at the state in which it ended for the previous state of the model results in the chain being initialised very close to the current state of the model. Results indicate that PCD leads to a model that more accurately learns to model the variance in the training data, yet results in a higher reconstruction error [21, 45, 2]. This would indicate that recreating samples from their RBM feature encodings would benefit from CD-1 whereas generating plausible unseen samples from the data set's underlying distribution would benefit from PCD.

2.7.4 Deep Belief Networks

Deep learning is concerned with creating a hierarchy of representation in order to represent complex data. At each consecutive level of the learned hierarchy, the representations become more abstract and constitute combinations of the lower layers representations in describing the data. A Deep Belief Network (DBN) is constructed by layering multiple RBM layers, connecting the output of a lower layer to the input of a higher layer. Bengio [1] outlines a greedy layer-by-layer training algorithm, utilising Contrastive Divergence, for training each RBM layer in a DBN. DBNs have been shown to achieve a level of hierarchical learning able to represent abstract representations of complex data [31, 29]. This hierarchical learning can be followed by other procedures, such as supervised back-propagation of weight derivatives, that fine-tune all of the weights to improve the generative or discriminative performance of the network as a whole [20].

2.7.5 RBMs, DBNs and Speech Data

There have been a number of research papers published around the application of DBNs to speech data. Mohamed et al. [36] probes the efficacy of training DBNs on MFCCs and a Fourier-transform-based log filter-bank (fbank) respectively. Additionally, the authors probe the performance of the DBNs extracted features for classification of spoken phones[11] as a function of the depth (number of layers) of the DBN. It was shown that the DBN performance was improved using fbank features as input as the fbank features better utilise the DBNs ability to discover higher-order structure in the input data. A phone classification error rate of 22.7% is reported.

The authors in Jaitly and Hinton [25] move away from using pre-processed and low dimensional input, and choose to instead determine whether a DBN can be trained on raw audio frames. This is important as success on such low-level and high-dimensional input would illustrate the ability of DBNs to successfully model some unknown latent variable in such a complex distribution. The authors report a phone error rate of 21.8%,

[11] Distinct speech sound.

competing with the state of the art with a minimal amount of pre-processing on the input.

In Deng et al. [8], the authors investigate an auto-encoder architecture that is initially pre-trained as a DBN. The pre-training phase is treated identically to any DBN approach: a greedy layer-by-layer generative unsupervised training in order to recreate the inputs. The network is then "unwrapped" to form a deep auto-encoder, and fine-tuned using standard supervised training with back-propagation of weight derivatives. The efficacy is probed in terms of log-spectral distortion of the recreation. Even with just the unsupervised pre-training to initialise the auto-encoder, a better recreation with lower distortion is achieved than without.

Work involving Convolutional RBMs note advantages of convolution in the analysis of high-dimensional data [32]. With this in mind, Lee et al. [32] makes use of an extended type of DBN which utilises convolutional RBM layers in order to learn a set of features that can be applied across spatially correlated points within the data. The argument for this is that if a useful feature can be learned from a patch of the data, then it can be useful to look for that feature across the whole data sample. This convolutional approach, combined with a sparsity regularisation[12] has been shown to produce features that are correlated with spoken phones when trained on speech data [32]. The definition of this network is integral to the work presented in this dissertation and is established further in Section 2.7.6.

All of the DBN architectures presented in this section made use of the TIMIT corpus and were tested via a phone classification task using the learned DBN extracted features as input to the classifier.

2.7.6 Convolutional RBMs

The Convolutional RBM (CRBM) is introduced in Lee et al. [31]. The CRBM differs from the traditional RBM in that there are a set of k weights shared among all locations in the input, rather than every visible node connected to every hidden node. k is a user-defined parameter and represents the number of learned features, or *bases*. The input layer consists of $N_{Vc} \times N_{Vw} \times N_{Vh}$ visible units, where N_{Vc} is the input channel depth, N_{Vw} is the input width and N_{Vh} is the input height. The hidden layer consists of a set of k *groups* of hidden units known as *feature maps*. Each feature map f_k is the result of convolution of the weight group W_k with the input v and has dimensions $N_{Hw} \times N_{Hh}$ where N_{Hw} is the width of the feature map and N_{Hh} is the height of the feature map. Each W_k is a convolutional filter with $N_{Wc} \times N_{Ww} \times N_{Wh}$ nodes, where $N_Wc = N_Vc =$ the channel depth of the input, N_{Ww}, or filter width, is defined to be $N_{Vw} - N_{Hw} + 1$ and N_{Wh}, or filter height, is defined to be $N_{Vh} - N_{Hh} + 1$. Additionally,

[12] A penalty that forces only a certain percentage of hidden units to be activated given any input.

all hidden units in a group h_k share a bias b_k, and all visible units share a single bias c. This convolutional approach allows smaller features to be learned across the whole image, which helps reduce the computation time when training on large input.

The logical starting point when studying any type of RBM is its energy function. It is from this energy function and the definition of the joint probability of a given combination of visible and hidden unit values $(\boldsymbol{v}, \boldsymbol{h})$ that the equations governing inference at the visible and hidden layers, and the weight updates, can be derived. Lee et al. [32] defines the energy functions for both binary and Gaussian real-valued input CRBMs. From this the equations for inferring the hidden layers given the visible layers and vice versa are derived as

$$P(h_j^k = 1|\boldsymbol{v}) = sigmoid((\tilde{W}^k *_v \boldsymbol{v})_j + b_k) \tag{2.74}$$

$$P(v_i = 1|\boldsymbol{h}) = sigmoid(\sum_k (W^k *_f \boldsymbol{h}^k)_i + c) \tag{2.75}$$

$$P(v_i = 1|\boldsymbol{h}) = Normal(\sum_k (W^k *_f \boldsymbol{h}^k)_i + c, 1) \tag{2.76}$$

It should be noted that 2.75 is used for Binary visible units, and 2.76 is used for Gaussian real-valued inputs. It should also be noted that the convention $*_f$ and $*_v$ represent *full* and *valid* convolutions respectively[13] and $\tilde{W}_k$ represents a 180 degree rotation of the weight group W_k about its centre.

Similarly to 2.71, 2.72 and 2.73 one can now define the parameter updates for the CRBM as follows:

$$W^k \Leftarrow W^k + \alpha(\boldsymbol{v}^{(t)} *_v rot180(\boldsymbol{h}^k) - \tilde{\boldsymbol{v}} *_v rot180(\tilde{\boldsymbol{h}}^k)) \tag{2.77}$$

$$b^k \Leftarrow b^k + \alpha(\sum_{i,j} (h_{i,j}^k - \tilde{h}_{i,j}^k)) \tag{2.78}$$

$$c \Leftarrow c + \alpha(\sum_{i,j} (v_{i,j}^{(t)} - \tilde{v}_{i,j})) \tag{2.79}$$

A Note on Sparsity

Noted that this convolutional model is over complete due to the fact that the representation is now larger than the size of the inputs. This arises as the output is now a set of k feature-maps with a size roughly equal to the size of the input. As over complete models tend to learn trivial solutions [31] it is important to force the representations to be "sparse".[14] This sparsity regularisation is outlined in Lee et al. [30] for non-

[13] Given an m-dimensional vector and an n-dimensional kernel (where $m > n$), valid convolution gives a $(m - n + 1)$-dimensional vector, and full convolution gives a $(m + n - 1)$-dimensional vector

[14] This involves a regularisation term on the hidden bias update that forces only a small fraction of the units in any given group to be active in relation to some input

convolutional RBMs and in Sohn et al. [43] for CRBMs and adds a sparsity penalty term to the hidden bias updates in order to promote sparse activity in the hidden units.

Probabilistic Max Pooling

The final component in the architecture presented in Lee et al. [31] is the pooling layer. In order to ensure a hierarchical model is learned, higher layers in the network should be operating on larger portions of the input. In traditional Convolutional NNs [28], this is done by a simple Max Pooling layer that shrinks a neighbourhood of the input by some scaling factor by only promoting the max value in that neighbourhood. The problem arises that this Max Pooling operation is deterministic, and thus does not support probabilistic bottom-up and top-down inference. The authors develop a probabilistic analogue to the Max Pooling operator that they creatively name Probabilistic Max Pooling. This operator restricts nodes in a neighbourhood of the hidden layer in such a way that at most one of them may be active at any time. This way, top-down inference may be done in order to recreate the RBMs input in a probabilistically sound way. For a more detailed discussion of this approach the reader is directed to Lee et al. [31].

Chapter 3

Implementation

3.1 Introduction

Performing the experiments and collecting the results that probe the question at hand was achieved through implementing a working CRBM. The implementation details, code libraries and data sets are introduced in this chapter. This serves to illustrate the scope of work that was self-implemented while crediting third-party libraries where used, as well as provide insight into the data sets that were used for verifying the implementation and running the experiments. Furthermore, a qualitative approach to verifying the correctness of the implementation is discussed.

3.2 Functional Point Breakdown

The implementation comprises of three modules: data pre-processing (if required based on input data type), unsupervised pre-training of the CRBM feature extractor, and verification of the learned features by means of a supervised classification task. The crux of the problem at hand is the CRBM feature extractor. This was implemented by the author with only Theano, discussed below, as a library for complex matrix operations and to port the hand written code to run on a GPU. This is important to point out, in the author's opinion, as it shows the scale of work that has gone into the development and verification of the CRBM as a technique for automated feature extraction, as well as demonstrate a working knowledge of the theory behind the technique. Furthermore, the supervised verification step was also hand developed as a framework within which multiple verification experiments can be performed with minimal configuration. The supervised classifier used in this verification framework was provided by a third-party library as it is merely a tool used to measure the efficacy of the feature extractor to extract useful features. This classifier can be swapped out with any other classifier as a step in the self-developed verification framework. Table 3.1 illustrates the scope

Table 3.1: Functional point breakdown

Functional Point	**Self Implemented**	**Third Party Lib**
Audio I/O & Fourier Analysis		librosa
Data pre-processing (Data standardisation, PCA decomposition)		scikit-learn
RBM layers + inference operations (Binary Visible & Real-Valued Visible)	X	
Sampling operations (Gibbs sampling & Persistent-Gibbs sampling)	X	
GPU backed matrix operations used in inference operations		Theano
Contrastive Divergence RBM layer training algorithm	X	
DBN model consisting of layers of RBM	X	
DBN grid-search experiment framework for hyper-parameter search	X	
Training statistics collection and visualisation	X	
DBN verification framework	X	
Stochastic Gradient Descent classifier used in verification		scikit-learn

of work that was implemented as a part of this project, and the portions that were provided by third-party libraries.

3.3 Language and Framework Choices

In order to effectively re-implement and verify the findings reported in Lee et al. [32], a programming language was employed that is well established, research-friendly and familiar to the author. Since the implementation deals with complex arithmetic and large matrix-based convolution operations, the language was also required to support optimisation libraries in order to reduce the training time to a level that allowed for training multiple models with various hyper-parameter settings. Based on these criteria, Python2 was chosen as the implementation language.

3.3.1 External Libraries Used

Librosa: Audio Signal Analysis

During the pre-processing of audio data, WAV file I/O and Fourier analysis functionality was provided by McFee et al. [35]. The library provides a `load(path, sampleRate)` function that takes in a `URI` path identifying an audio `.WAV` file, along with a desired sample rate, and returns an array of time-series sound pressure readings. Additionally, the library provides STFT functionality that operates on this array of time-series sound pressure readings. The `librosa.core.stft` function takes, along with the time-series

audio data, options for the number of samples to consider in a single DFT window, `n_fft`, and the number of samples to move the window forward by before taking the next DFT. The result of this operation is a 2-dimensional matrix of complex values such that the the absolute value of entry `(f, t)` is the magnitude of the frequency-bin `f` at time-frame `t`, and the counter-clockwise angle from the positive real axis on the complex plane of each entry `(f, t)` is the phase of the frequency-bin `f` at time-frame `t`.

NumPy/SciPy

Array creation and manipulation operations as well as I/O of post-processed data arrays was provided by NumPy [38].

Theano and CUDA

Theano is a library that allows for the definition of complex mathematical expressions as a computational graph. This graph can then be compiled into fast-running and efficient C-code. Through a tight integration with Nvidia's CUDA GPU programming library [39], as well as CuDNN for efficient deep neural network operations, Theano backed code can be easily ported to run on a GPU back-end, thereby dramatically decreasing the execution time of neural network training [44].

Scikit-learn

Scikit-learn's [41] library of machine learning algorithms was used during the pre-processing phase of the audio data as well as the verification phase of the overall implementation. For pre-processing of audio data, after taking the STFT, the Principal Component Decomposition algorithm was utilised to reduce the dimensionality of the magnitude spectrum from 160-channels to 80-channels. The `linear_model.SGDClassifier` implementation of the Stochastic Gradient Descent algorithm was employed in the supervised classification verification phase. Scikit-learn was chosen as it has been widely accepted in scientific computing, and is built on NumPy and SciPy. This model provides a `partial_fit` method that, when invoked, will perform curve fitting with SGD or Batch-GD. This method is provided with a batch of training values, and an array of target classes for each training value in order to perform the training. Once trained, the same model provides methods to predict class label confidence for a given batch of test data samples.

3.4 Datasets

MNIST

As with many machine learning implementations, the initial development was tested on the MNIST handwritten digit database [28]. The data set consists of 50 000 black and white handwritten digits of shape 28x28 pixels. Using this data set allows for minimal pre-processing, fast execution as the input size is small, and since the input are images the results, and thus correctness of the implementation, are able to be visually verified. While visual verification of the network's recreation of an input and its learned filters does not guarantee that useful feature extraction has been learned, a successfully trained network will behave as so. Failing to do so indicates either an incorrect implementation or a poor selection of hyper-parameter values [21].

TIMIT

As this project's aim was to re-implement experiments from Lee et al. [32], the TIMIT data set was used as it is the same data set from the paper and promotes reproducibility of the result. TIMIT was designed to provide speech data for the acquisition of acoustic-phonetic knowledge and for the development and evaluation of automatic speech recognition systems [14]. TIMIT contains a total of 6300 sentences, 10 sentences spoken by each of 630 speakers from 8 major dialect regions (DR1 - DR8) of the United States. The data set consists of roughly 70% male utterances and 30% female utterances, and contains three different types of sentence construction. Two dialect sentences (labelled "SA" sentences) were designed to expose the dialectal variants, and were spoken by all 630 speakers, while 450 phonetically compact sentences (labelled "SX" sentences) were designed to provide a good coverage of pairs of phones. Each speaker read 5 of these "SX" sentences, and each "SX" sentence was read by 7 different speakers. Finally one last category of sentences exist to be phonetically diverse (labelled "SI" sentences). These were chosen from existing text sources to add diversity in the sentence types and phonetic contexts. A tabular breakdown of the dataset can be seen in Table 3.2. Furthermore, TIMIT provides full contextual and phonetic transcriptions of the sentences spoken, including the start and end position of the spoken phones. The same data set segmentation from Lee et al. [32] was chosen for training and verifying the classifier to ensure that identical audio samples were used.

3.5 Implementation Verification

Successfully training an RBM is a difficult task. Even more so when developing the code from first principles. It is necessary to run many training iterations and verify whether

Table 3.2: TIMIT sentence breakdown

Sentence Type	**#Sentences**	**#Speakers**	**Total**	**Sentences per Speaker**
Dialect (SA)	2	630	1260	2
Compact (SX)	450	7	3150	5
Diverse (SI)	1890	1	1890	3
Totals	**2342**		**6300**	**10**

or not the internal inference and training algorithms have been correctly implemented. Furthermore, one needs to decide how to set the many hyper-parameters that are utilised within the training algorithm, which types of units are best suited for the given data type and what input-batch size, if any, to use. This type of practical guidance was provided by Hinton [21].

The quantitative method of testing the richness of features an RBM model has learnt is to test how well they perform in a classification task. With that being said, there are a few metrics that can be collected during the training process that indicate whether the negative log-likelihood function is converging to a minimum or diverging, and whether the learned filters extract useful features.

Reconstruction Error

The negative log-likelihood that is being minimised during training emphasises the difference between the expectation of the network under a measured data point, and the expectation of the network under a sample from the current state of the model. When training via a single step of Gibbs sampling (Which is also known as CD-n, with $n = 1$), the emphasis is on the difference between the data point expectation, and the expectation of the network's recreation of the data point. It can therefore be noted that the difference between a data point and the network reconstruction should, in general, decrease over the course of training. To monitor this reconstruction error, and how it changes over time, we periodically collect it according to the following equation:

$$Err_{reconst} = (x_{original} - x_{recreation})^2. \tag{3.1}$$

Squaring the difference between the original input $x_{original}$ and the network recreation $x_{recreation}$ ensures that we are always dealing with a positive value. It is noted in Hinton [21] that while this measure is convenient, it can, in reality, be a poor indicator of the progress of learning, especially for CD-n with $n \gg 1$. This is due to the fact that the reconstruction is not the function that the learning is optimising, as the reconstruction is not a true sample from the model's distribution, which is what is actually being optimised. It is noted, however, that the reconstruction error should fall, and that

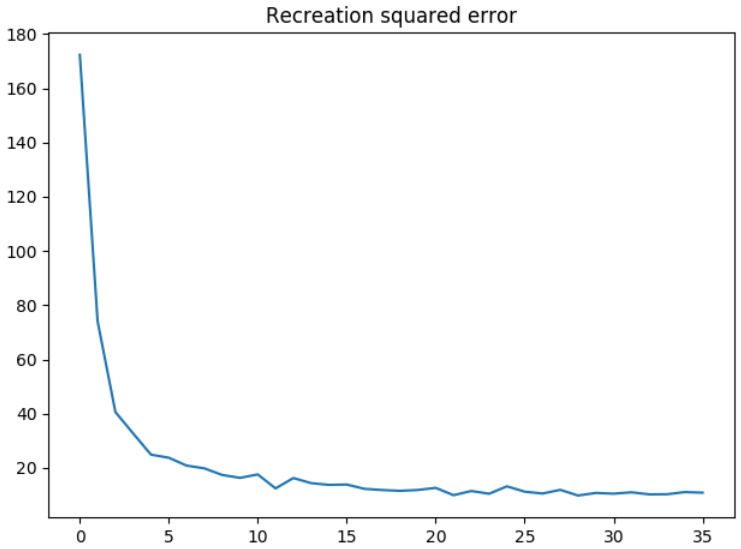


Figure 3.1: Recreation squared error collected while training a binary-binary CRBM layer on MNIST data.

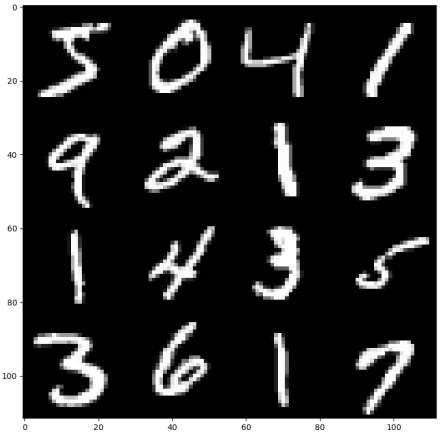
large increases are a sign of divergence [21]. Figure 3.1 shows the reconstruction error over the course of training of a CRBM layer on the MNIST data set. Figure 3.2 shows an example of input and recreation comparison.

Weight and Bias Histograms

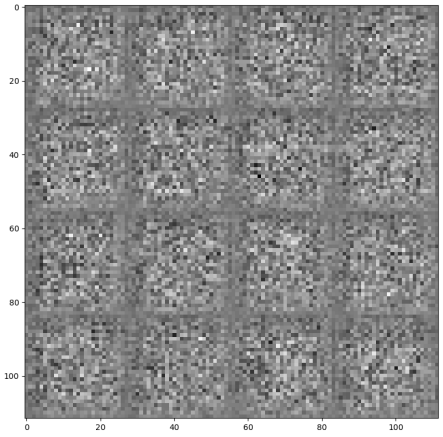
Histograms of the RBM layer's weights and biases along with increments added to them when being updated can be very useful for diagnosing common problems during training. Hinton [21] notes that if the learning rate is set appropriately the update increments should be roughly 10^{-3} of the value of the weights or the biases respectively. Values smaller than this may indicate a learning rate that's too slow, and values that are larger may lead to weight and bias values growing too large such that the training process does not reach an equilibrium. When divergence occurs the histograms will show this happening, and the training can be stopped. Collecting these stats, however, can be expensive and should not be done at every update. Figures 3.3 and 3.4 show an example of the histogram data collected for the weights and the hidden biases respectively during a training instance on the MNIST data set.

Weight and Feature Map Visualisation

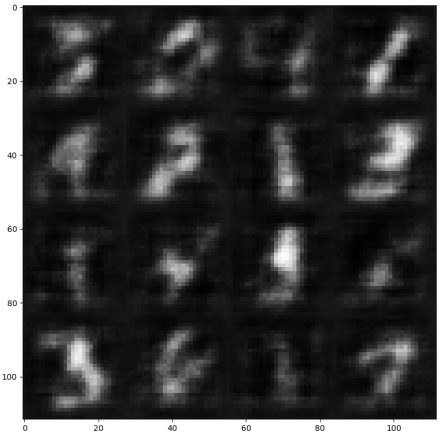
Finally, for domains in which the input units have spatial or temporal structure, such as images or audio spectrograms, it is often useful to visualise both the filters being learned (the weights) as well as the extracted features (the hidden unit activations). By visualising the filters, especially in the first layer of an RBM trained on data that can be visualised, we can get a good idea of what type of features have been learned.



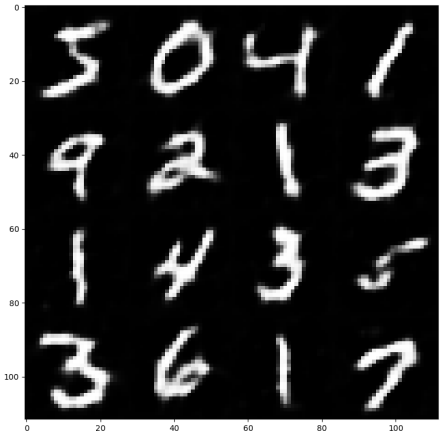
(a) Original batch of 16 inputs.



(b) Untrained network recreation.



(c) Network recreation mid-training.



(d) Final network recreation.

Figure 3.2: MNIST input and recreation comparison.

An example of this can be seen in Figure 3.5. This visualisation shows horizontal and diagonal lines, as well as dark spots and points, being learned features of the MNIST data set. Visualising the feature maps at the hidden output nodes can also indicate whether or not some hidden units are never used or if some training samples activate an unusually large or small number of hidden units. This can guide the network architect in the setting of the sparsity controlling hyper-parameters. Figure 3.6 shows a visualisation of the feature maps extracted from an MNIST training sample. This is the information that the CRBM implementation learns to extract. Looking at Figure 3.6, it looks to the author that each feature map is a different representation of the original input, breaking down information about the input into multiple different views and providing more information than the pure raw data.

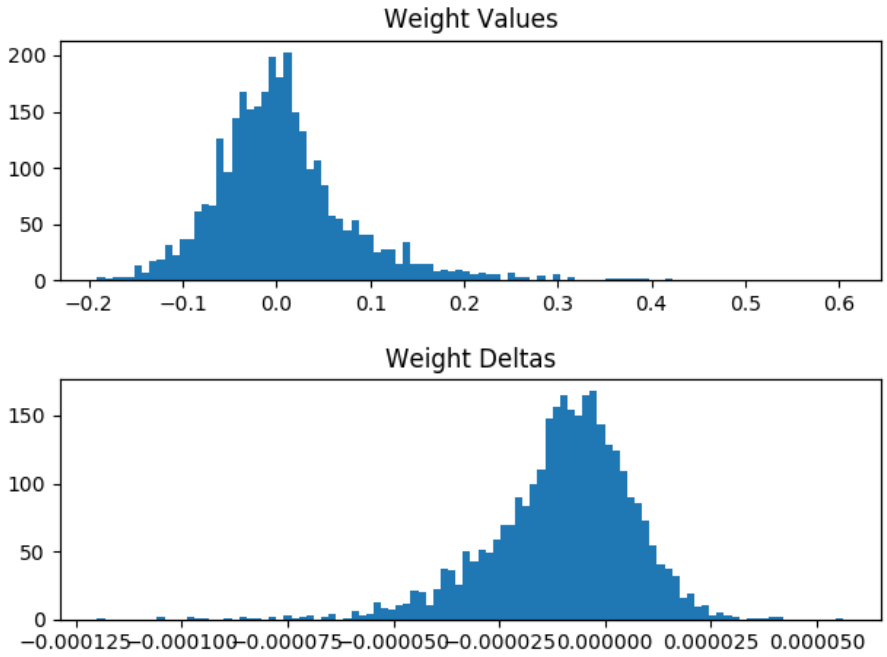


Figure 3.3: Histogram of network weights and update values during training on MNIST data.

Initial Values

For the initial untrained values of the hidden and visible biases, as well as the weights, were set in accordance with Hinton [21]. The biases are initially set to 0, and the weights are randomised according to the normal distribution with a variance of 0.01.

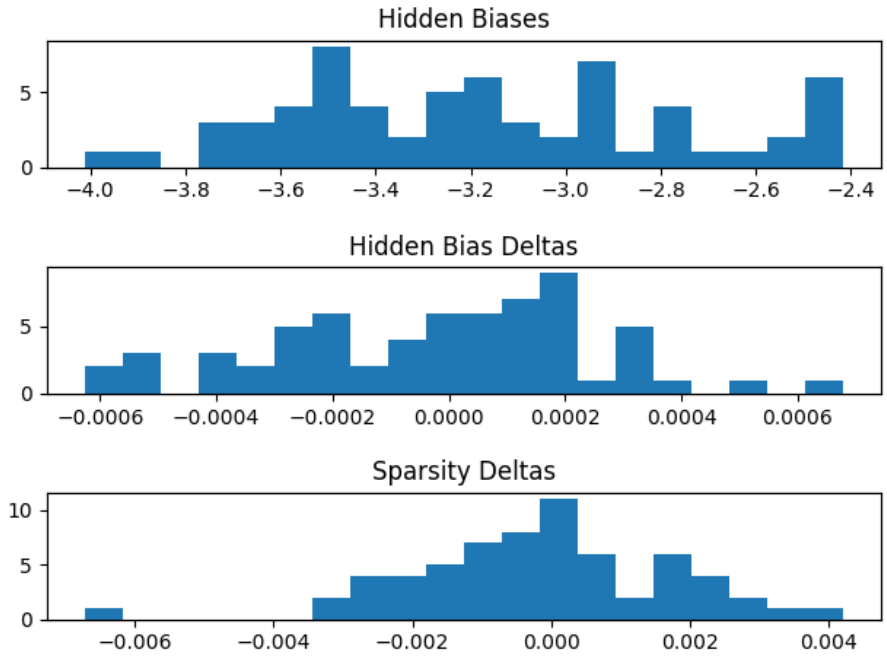


Figure 3.4: Histogram of network hidden biases and update values during training on MNIST data.

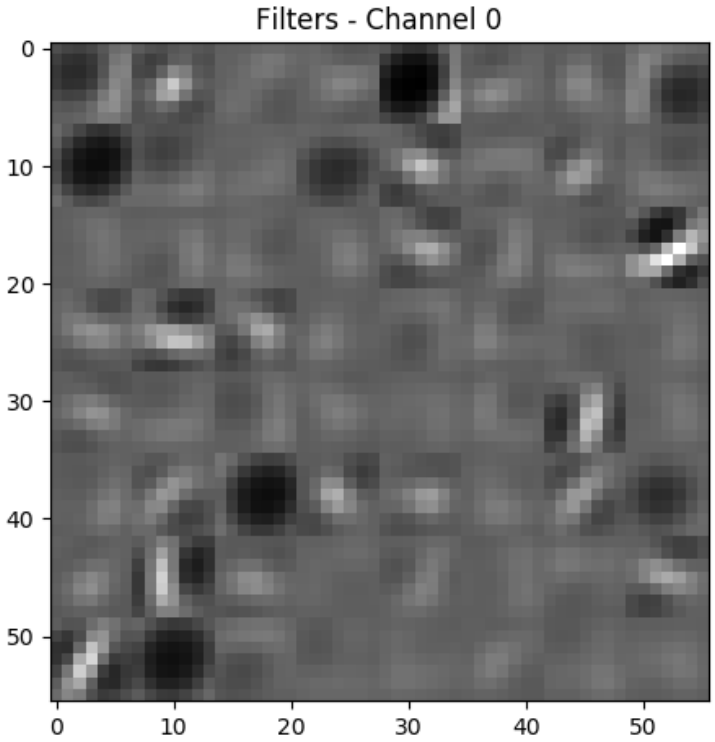


Figure 3.5: 64 filters (8x8 grid) learned by binary-binary CRBM on MNIST data set.

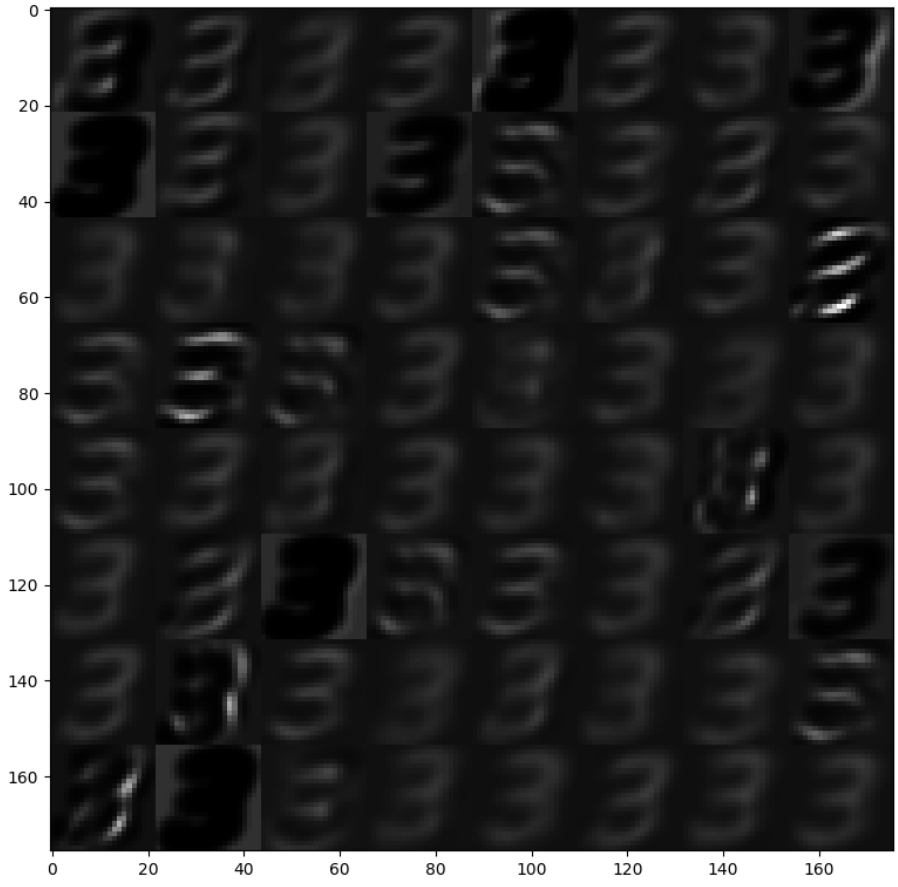


Figure 3.6: 64 feature maps generated by the trained binary-binary CRBM.

Chapter 4

Experimental Set-up

4.1 Introduction

With a working CRBM implementation in hand, an experimental approach must now be designed in order to test the efficacy of the implementation to extract useful features from speech data. The sections in this chapter cover how the available speech data set and CRBM implementation are used to set up, perform and verify the results of such an experiment.

4.2 Pre-processing Pipeline

Before presenting any given sample from the TIMIT data set to the CRBM implementation, it needs to be pre-processed. This pre-processing performs the transformations, standardisations and dimensionality reductions that we desire before training. The full pipeline is illustrated in figure 4.1.

STFT

The STFT settings used for pre-processing were chosen in accordance with Lee et al. [32]. This involved a sample-rate of 16000Hz, a Hann window with an FFT size of 320 samples, and a window hop-length of 160 samples. This results in an STFT output which is 160 frequency-channels deep. Before reducing this, the data needs to be standardised.

Figure 4.1: Pre-processing pipeline for audio samples.

Standardisation

There are a few ways that sets of features can be standardised in order to be compared. This is usually done on features that are measured at very different scales, but also when the values of the features over the data set have very different means and standard deviations. By removing each feature's mean and scaling by its standard deviation, we get a more standardised view of how the features deviate from their average point, which will now have been made the same for each feature. This was specifically done before learning the PCA model as it was noted that without this step the first principal component would learn the mean within the data, which would ultimately dominate the features learned by the CRBM layer. The mean and standard deviation were learned for each of the 160 frequency channels across the entire TIMIT training set.

PCA Whitening

After removing the learned mean from a given sample and rescaling by the learned standard deviation, it can be reduced to its principal components using the Principal Component Decomposition model. The model was trained on the full TIMIT training set after it had been standardised. The model was constrained to utilise the first 80 principal components, and as per Lee et al. [32] included a final "whitening" step. This final step divides the extracted components by the square root of their related eigenvalue, thereby ensuring that the different components are uncorrelated and have unit variance, which is important in simplifying the inference procedure in a real-input valued CRBM layer [21, 31]. After the PCA decomposition step, the input is reduced from 160 channels down to 80 principal component channels.

4.3 Unsupervised Pre-training

The unsupervised pre-training of the CRBM to perform meaningful feature extraction is at the core of this dissertation. It is the efficacy of these features to represent information in speech data that the following experiments aim to assert. Due to the implementation, each training input in the learning phase is required to be of the same size. The channel depth of the samples is inherently the same due to the sample rate, the STFT parameters and the PCA model (limited to 80 principal components). The temporal length of the samples, however, needs to be made consistent. After analysing the TIMIT training set, the number of time-steps per training sample was chosen to be 92. This was the length of the shortest training sample. Longer samples were broken up into sub-samples of 92-time steps in length.

This pre-processing results in a 1-dimensional input vector of size 92 with a channel-depth of 80. Once again the remaining hyper-parameters of the 1-D filter width N_W and

the pooling ratio R_{pool} were set in accordance with Lee et al. [32] to ensure comparable results. The values were taken as $N_W = 6$ and $R_{pool} = 3$ respectively. For the gradient-descent training routine, a mini-batch size of 16 was chosen as it is a power of two large enough to take advantage of concurrent operation speed-up, but small enough that the gradient steps are not overly smoothed out [1, 32]. Visualisation of the learned filters after training on the pre-processed TIMIT training set is presented in Chapter 5.

4.4 Feature Extraction

Once the CRBM has been trained it can be used to extract features as inputs to another algorithm (like a classifier model). To extract the features for a given piece of data, it first goes through the pre-processing pipeline and is then propagated through the CRBM network. The hidden unit activations, or feature maps, are then taken as the extracted features for that layer. In the case of a layer that makes use of Probabilistic Max Pooling, the pooled layer activations are used as the extracted feature maps. For the presented application and experiments, the feature maps are then unwrapped to form a single 1-dimensional feature vector. This is done as the classifier models do not take spatial placing of features into account, but rather look at every pixel simultaneously. What's important is that the unwrapping of the feature maps to a 1-dimensional feature vector be done in a consistent manner each time.

4.5 Feature Verification

In order to test the efficacy of the features that the CRBM implementation learns to extract in a concrete and empirical manner, we train a standard linear classifier algorithm while utilising the extracted CRBM feature vectors as input. The classifier in question is a logistic regression model trained using SGD, using a log-loss function, by Pedregosa et al. [41]. This algorithm was chosen as using a logistic regression loss function yields a form of multinomial logistic regression, which enables the ability to predict confidence across multiple class labels. This type of regression model is a generalized linear model, chosen to align with the experimental set-up from Lee et al. [32], the exact implementation of which is unknown to the author. The classifier is trained in a supervised fashion and ultimately tested with a portion of data that has not been shown to either the CRBM feature extraction layer or the classifier. This is done to determine whether the classifier has learned a policy of classification that extends outside of just the training data that it learned from. This approach follows from the verification experiments performed in Lee et al. [32].

Training

To prepare a given training datum for the classifier it is first passed through the pre-processing pipeline and then split up into equal sub-samples of T time-frames in length, where T becomes a tunable hyper-parameter of the classifier model. Each sub-sample is assigned the same class label as the full sample. The sub-samples are then passed through the CRBM feature extractor and feature vectors are generated for input to the linear classifier, along with the corresponding class labels for training.

Classification

Once trained, the multinomial logistic-regression classifier can be tested on the reserved test set. Much in the same way as the training step, each test sample is pre-processed and divided into T equal sub-samples and for each instance, feature vectors are generated by the CRBM. The learned linear classifier is now asked to predict the confidence percentages of the available class labels for the sub-samples. Each sub-sample then provides a weighted vote for it's most confident class label, weighted by the actual confidence percentage. In this way, if a sub-sample's highest predicted class label has a high associated confidence percentage, it will vote with a higher weighting toward the overall label for the full sample. A final prediction for the whole sample is then arrived upon by summing all the sub-samples' weighted votes for the target label. This prediction can be compared against the ground-truth target vector to determine the classification success or error rate.

All experiments performed were carried out with a single CRBM layer. This was done due to a lack of sufficient computing power and to complete within planned time constraints.

Chapter 5

Results

5.1 Introduction

With the implementation and experimental approach defined, experiments were carried out and results collected. Two experiments in particular were performed, and were done so in the same manner and on the same data set as outlined in Lee et al. [32]. Specifically, these two experiments probed the ability for the extracted features to be useful in a genetic sex classification task, and a speaker identification task. This chapter introduces the detail around the experiments, as well as the results collected. These results are compared to the results previously collected and reported in Lee et al. [32]. Furthermore, results collected when testing the implementation's *generative* abilities are presented for two different CRBMs that were trained with single-step CD and persistent CD respectively.

5.2 Classification Results

A brief overview of the results from both classification experiments is presented in Table 5.1. It is interesting to note that the best performance in both of the classification experiments came from RBMs trained with the CD-1 sampling procedure. RBMs trained with PCD did not show any marked improvement. If anything, the features extracted with the PCD trained CRBM performed marginally worse in the classification verification step. The discrepancy between results for speaker identification is discussed in Section 5.2.2.

Table 5.1: Results from classification experiments.

Experiment	**Error Rate**	**Result from Lee et al. [32]**
Genetic Sex Classification	3.77%	5.4%
Speaker Identification	17.14%	0.3%

5.2.1 Speaker Genetic Sex Classification

To probe the ability of the extracted CRBM features to represent information useful to classify genetic sex from speech utterances, a binary classification task was performed. For each sample in the training set, a target class of either 'M' or 'F' was provided during training representing genetic sex of Male of Female respectively. During test set verification, a class of 'M' or 'F' was predicted for a given sample and the prediction compared to the ground truth class label. The full TIMIT training/test set split was utilised for the experiment. This split involves 4620 training samples and 1680 testing samples, each with a genetic sex split bias of 70% male speakers to 30% female speakers. The classifier was found to need very few epochs to converge to a final result. An example plot of the overall classification error rate can be seen in Figure 5.1. A total of 10 randomly initialised training runs were executed, and a final classification error result was achieved by averaging the final classification error rate of each run. By doing so, we arrived at a mean classification error rate of **3.77%** with a standard deviation of 0.32%.

This result is conclusive. A policy was learned by the classifier that is reliably able to map the extracted CRBM feature representation to the appropriate class label 96% of the time. It is useful to compare gathered results to the most effective naive classifier. For the above unbalanced training and test set of 70% male speakers and 30% female speakers, this would be a policy that consistently classfies any sample presented to it into the 'Male' category. This would result in a classification error rate on the test set of 30%. The result of the learned model achieving an error rate of 3.77% offers concrete evidence that training has indeed been successful and that the model has learned a policy that far out performs the best naive model for the provided data set.

5.2.2 Speaker Identification

A second experiment was carried out in which the extracted features are tested on a 168-way classification task, where test utterances must be correctly matched to the speaker label of the person that the utterance is from. This means that during training there are 168 class labels, each one representing a specific speaker from a set of speakers. In order to build the training and test set for the 168-way classification, the same approach was followed as in Lee et al. [32][1]. This resulted in sets containing 112 male speakers and 56 Female speakers, where the training and test sets contained 8 and 2 utterances per speaker respectively. For this subset of training and testing data, the classifier was trained and validated over the course of 100 epochs. An example plot of the overall classification error rate can be seen in Figure 5.2. A total of 10

[1]For each speaker, we used eight training utterances (2 sa sentences, 3 si sentences and first 3 sx sentences); the two testing utterances were the remaining 2 sx sentences per speaker.

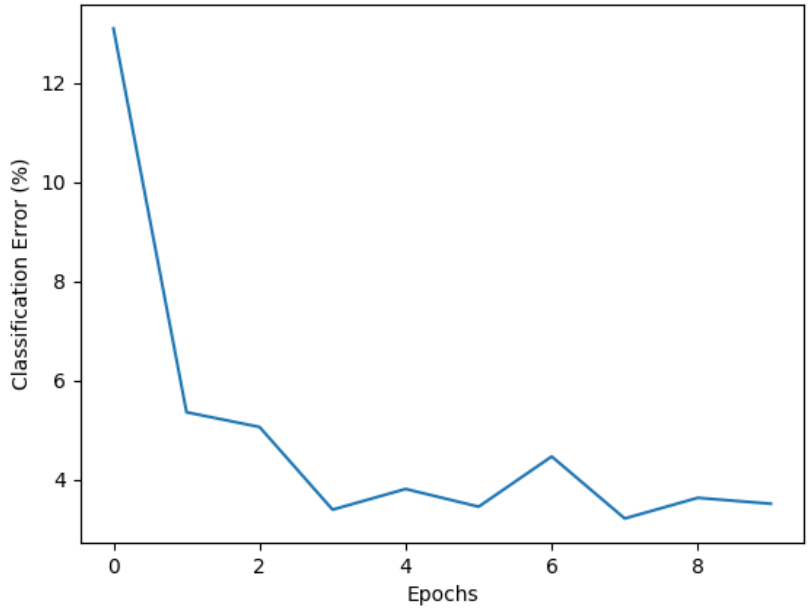


Figure 5.1: Classification error percentage as a function of epoch count (Genetic Sex Classification)

randomly initialised training runs were executed, and a final classification error result was achieved by averaging the final classification error rate of each run. By doing so, we arrived at a mean classification error rate of **17.14%** with a standard deviation of 1.15%. In comparison, the error rate reported by Lee et al. [32] is 0.3%. This discrepancy between gathered results and those reported in the paper of interest is believed to be due to some unreported complexity in the implementation presented by the authors.

While this result does not map directly to the results achieved in Lee et al. [32], it still indicates that a fairly accurate classification policy (albeit sub-optimal in comparison with the paper of interest) was learned. Once again to illustrate this, one can consider a purely random classifier. Provided the random classifier has a policy to choose equally randomly between any of the 168 speakers in the test set, then the likelihood of guessing correctly when presented with any random sample is $\frac{1}{168} = 0.006$. This means that a random policy classifier has a 0.6% chance of guessing any given test set sample correctly. We would therefore expect to see an error rate of around 99.4% over the whole test set when guessing randomly. Bringing this down to 17.14% is a drastic improvement and shows how successful the learned policy is.

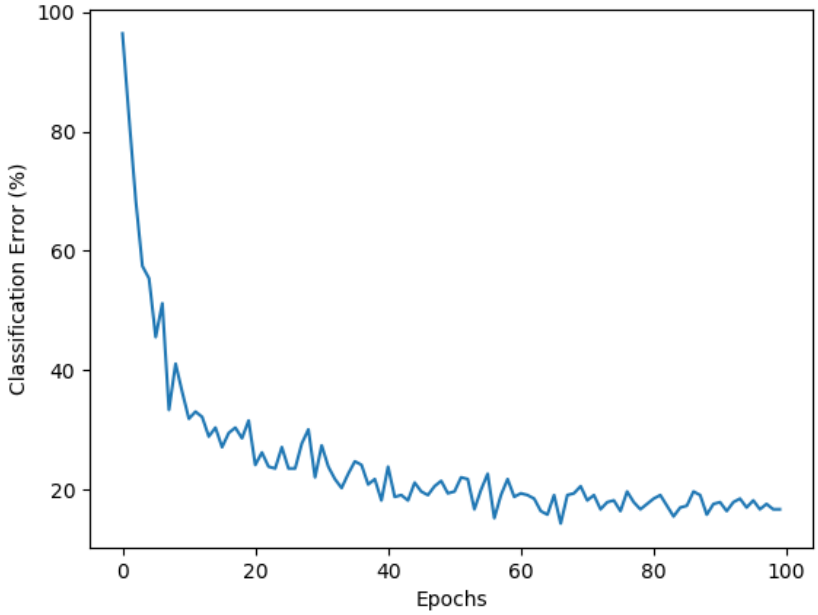


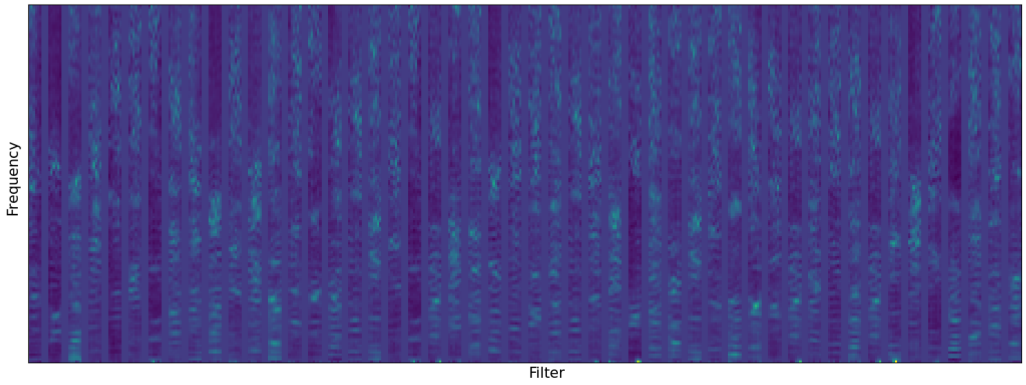
Figure 5.2: Classification error percentage as a function of epoch count

5.3 Visualising What Was Learned

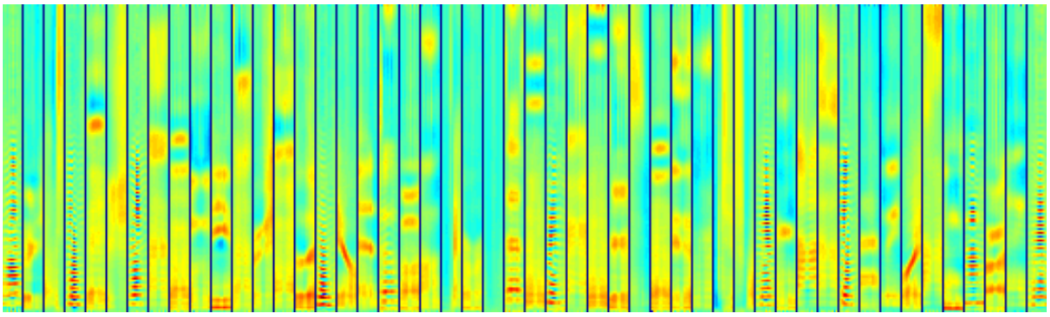
One way to inspect what was learned during the training process is to visually inspect the filters. We can do this by taking any number of the learned filters, and multiplying them by the inverse of the PCA decomposition step outlined in Chapter 4. A comparison of the filters learned from the author's implementation and those presented in Lee et al. [32] can be seen in Figure 5.3.

5.4 Analysis Through Synthesis

Referring back to Equation 2.70 one is reminded that the fundamental equation that we are minimising when training the CRBM, the negative log-likelihood, emphasizes the difference between the expectation of the CRBM under a given training sample, and the expectation of the CRBM under an actual sample from the model. Therefore, during training the network learns to create samples that should fit into the probability distribution described by the data set. We can therefore, theoretically as opposed to empirically, probe what the network has learned by means of visualising and synthesising samples from the model.



(a) 50 randomly selected filters (Each representing a 160x6 "temporal receptive field" [32]) learned by Gaussian-binary CRBM training on TIMIT data set.



(b) 50 randomly selected filters from Lee et al. [32].

Figure 5.3: Comparison of collected learned filters to those reported in Lee et al. [32]. For both (a) and (b), the y-axis represents the frequency channel, ordered lowest (bottom) to highest (top). Each column is a receptive field across all 160 frequency channels spanning 6 STFT time bins. The images are best viewed digitally and in colour.

5.4.1 Single-Step Contrastive Divergence

When single-step Contrastive Divergence (CD-1) is utilised as a method of sampling from the model, the sample is taken simply as the network's recreation of a given data point. Once again, as mentioned in Hinton [21], this is not a hugely accurate estimation of a true sample from the network, but does prove effective in training an RBM for feature extraction. Figure 3.1 has already shown how the recreation error drops when training with CD-1 on the MNIST data set. Figure 5.4 visually shows the ability for the trained CRBM to recreate a sample input from the TIMIT data set. The synthesis of this result to a final audio file was also done, a link to which can be found in Section A.1.

In order to get a better representation of the model, and to draw a more accurate sample, one can take the CRBM that was trained using CD-1 and initialise it with some uniformly random input. We then run a Markov chain of repeated Gibbs sampling for a certain number of steps, inferring and sampling forward and then backward. If we

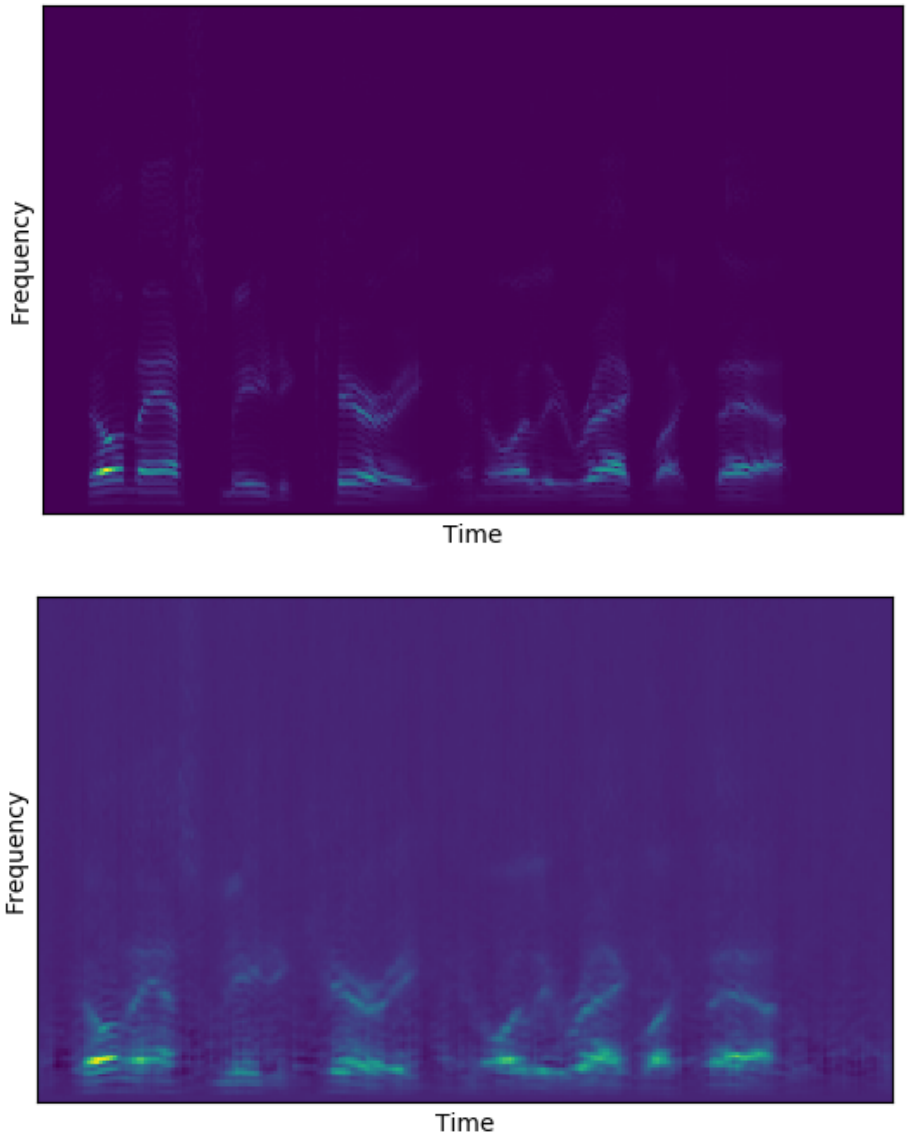


Figure 5.4: Original and recreated audio magnitude spectra for a random TIMIT sample

ran this chain for an infinite number of steps, we get a true sample from its underlying learned distribution. Figure 5.5 shows the result of 300 steps of Gibbs sampling initialised at a uniformly random starting point. This result was also synthesised as a final audio file to audibly inspect the result. The link to the result can be found in Section A.1.

5.4.2 Persistent Contrastive Divergence

When Persistent Contrastive Divergence (PCD) is utilised as a sampling strategy during training, the sample drawn from the model for the negative phase of the negative log-likelihood over time becomes closer to the model's underlying distribution. In this way, the network is not trained to exactly recreate a given training example, but to rather

update its internal representation in such a way that samples drawn from it closely match the distribution described by the data set. Figure 5.6 shows a sample drawn from a CRBM trained using PCD as the sampling strategy for the negative phase. One can quantitatively compare the audible difference between the synthesised sample from the CD-1 trained network and the synthesis of the PCD trained network. The synthesised results can be found in Section A.1.

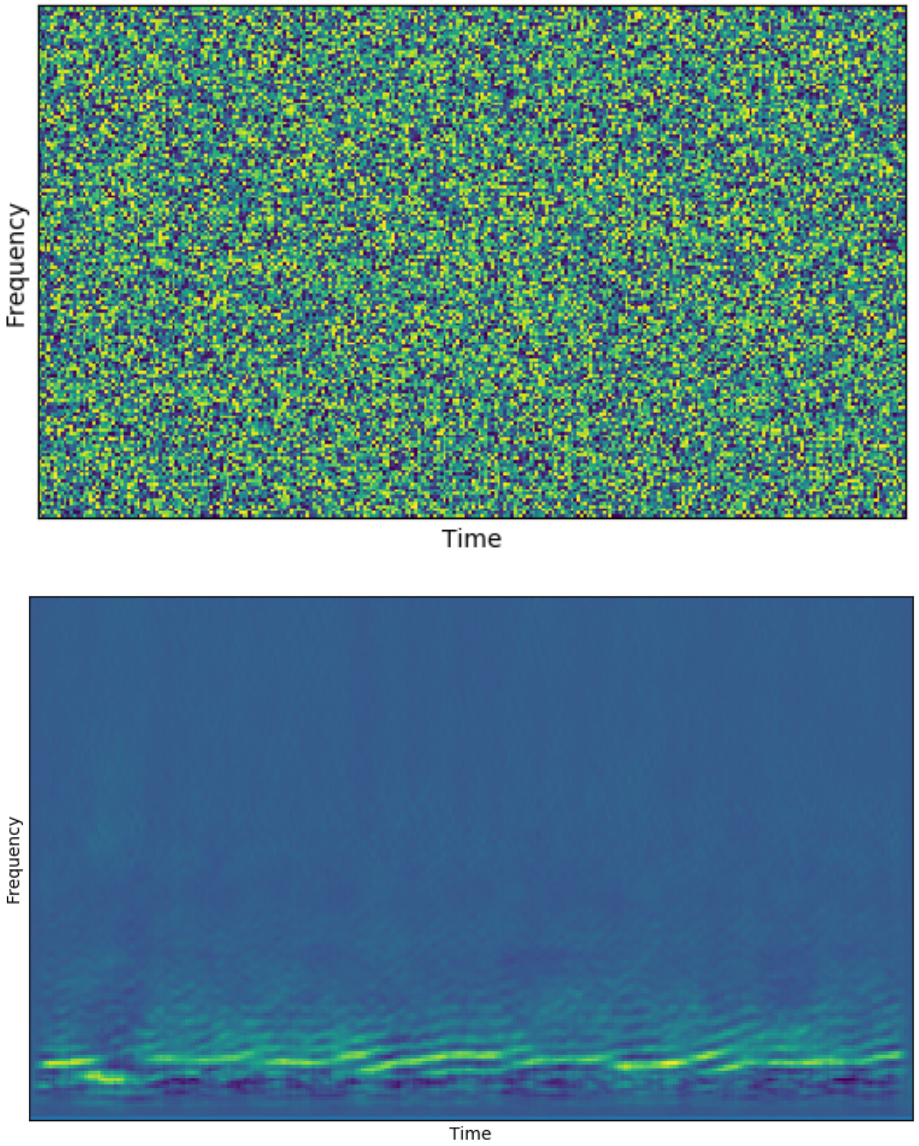


Figure 5.5: Initial Markov chain starting point and sample after 300 Gibbs sampling steps. Sampled from CD-1 trained CRBM.

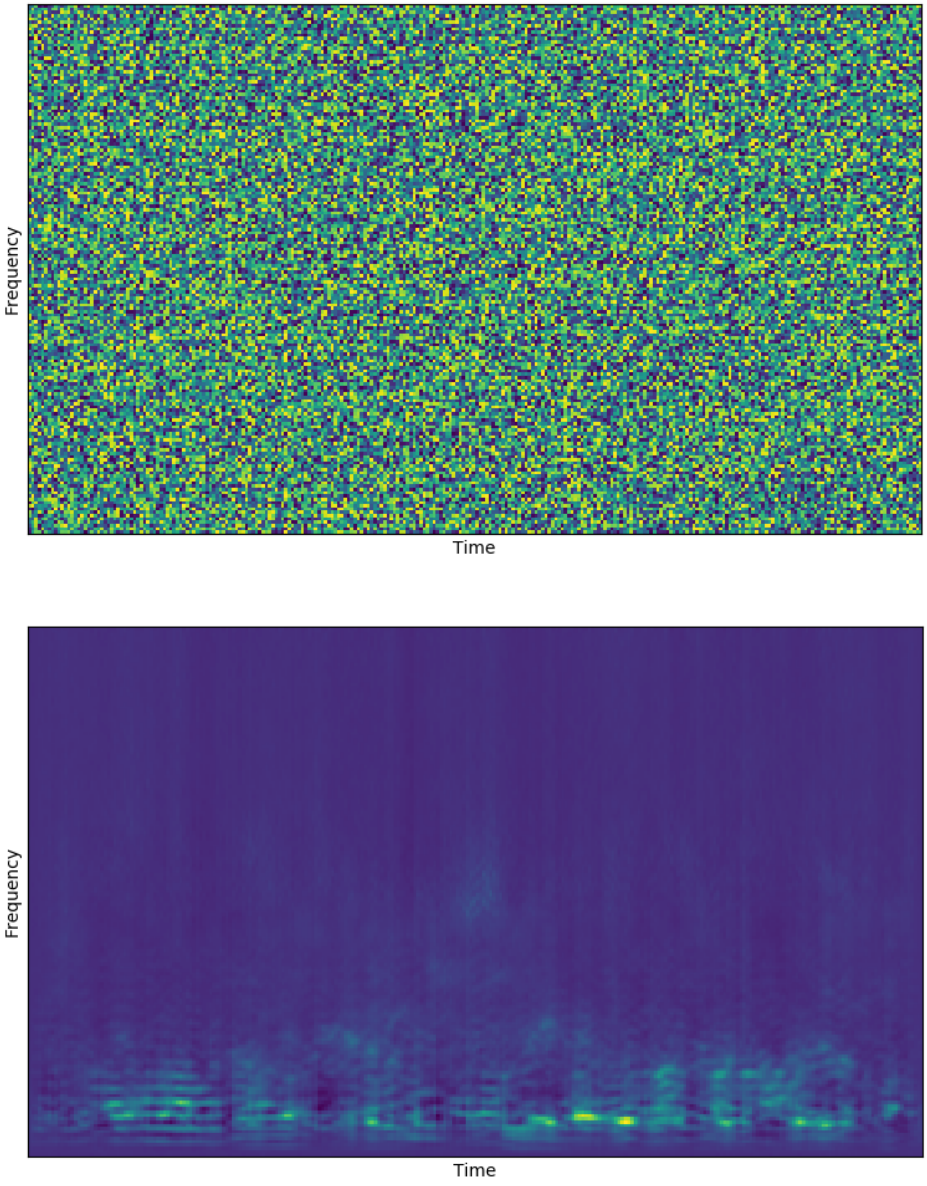


Figure 5.6: Initial Markov chain starting point and sample after 300 Gibbs sampling steps. Sampled from PCD trained CRBM.

Chapter 6

Discussion

6.1 Introduction

The results from the performed experiments were presented in Chapter 5. This chapter deals with breaking down what the results mean and how they relate to the problem at hand. Similarities and differences with the related work are discussed, and implications which may lead to related future work are presented.

6.2 Classification Results

The final results collected for the genetic sex classification problem and the 168-way speaker identification task yielded classification error rates of 3.77% and 17.14% respectively. The classification success rates for the two experiments are therefore 96.23% and 82.86%. The better performance in the genetic sex classification problem is an expected result as a binary class separation is easier for a classifier to learn than a 168-way separation which is more complex and has an innate higher probability of false classifications. This offers concrete evidence that the features the CRBM implementation learns to extract encode information useful for speech classification tasks in a compact and separable manner. This conclusion agrees with and reinforces the results from Lee et al. [32]. The authors of the paper report genetic sex classification success rates of 94.7% from a single layer CRBM experiment, which aligns closely with the results reported here. They report results from the 168-way speaker identification task with a classification success rate of 99.7%. This does not completely align with the presented experimental outcome. It is worth noting that it is strange that the results reported by Lee et al. [32] for the 168-way speaker identification task out perform their results for the binary classification task of genetic sex, especially considering the higher complexity and innate probability for more false classifications.

It is possible that the results of the speaker identification task could be somewhat

Table 6.1: Hyper-parameter settings of final trained CRBM model.

Hyper-parameter	**Value**
Learning rate	0.001
Pooling ratio	3
Sparsity target	0.05
Sparsity learning rate	0.1
Num learned filters	300

improved through a fine-tuning of the model's hyper-parameters. The values used for them in the final experiments, however, were determined through a grid-search optimisation routine where various 'grids' of parameters are tried and compared. At each step of the grid search the performance of the hyper-parameters was verified with the classification task. For reproducibility and transparency, the values for the various hyper-parameters are listed in table 6.1. Further fine-tuning the network's weights could also be performed in a manner described in Hinton and Salakhutdinov [23]. In this method, the CRBM would be 'unwrapped' to form an autoencoder. This autoencoder could then be trained using standard NN back-propagation in order to minimise the difference between the input and the network's output recreation. Results have shown that initialising an autoencoder in this manner can greatly increase success and speed of autoencoder training [23]. It would be worth seeing if this fine-tuning would result in an improved classification rate.

6.3 Voice-To-Text Possibilities

Lee et al. [32] defines each filter as a "temporal receptive field" in the spectrogram space. In other words each filter will be receptive to any temporal portion of the input data that highly correlates with it. As such, the forward-propagation, or feature extraction operation, of an input sample with a given filter will yield a feature map that emphasizes the temporal points that correlate with that filter. It is not hard to imagine that generating feature maps defining where (temporally) certain phones, or sound parts of spoken words, occur in a given input sample is a stepping stone for the development of a voice to text framework.

6.4 Generating New Samples

One aspect of RBMs that makes them interesting is their ability to act as a *generative* model. As mentioned in Section 2.7, RBMs are able to draw samples from the underlying joint-probability distribution $p(\boldsymbol{v}, \boldsymbol{h})$ learned from the training data. While sampling from this distribution is in general intractable, we have also seen from Section 2.6.4 that sampling procedures have been developed to overcome this. We have shown

in chapter 5 that samples that hold visual similarities to speech spectrograms can be sampled from the model with Gibbs sampling initialised completely randomly. This shows that the network has definitely had some measure of success in modelling the TIMIT speech corpus. This generative capability introduces some new possibilities. One such avenue worth investigating is that of signal recreation. It's feasible that initialising a model, trained on speech data, with a lossy speech data signal, and then performing a few iterations of Gibbs sampling, could result in a more accurate representation of the initial signal sampled from the model. Doing so could thereby fill in the lost portions of the speech sample and de-noise the signal, much in the same way as the practical example of Markov random fields presented in Section 2.6.2. It would also be interesting to adapt the current model to take a textual transcription of the input sample as additional input. It should then be theoretically feasible to condition the model on a given textual transcription and to generate speech samples with that transcription's spoken content. Much adaptation to the network and training would be needed in order to increase its temporal resolution over time, perhaps in a manner similar to Oord et al. [40].

Referring to Section A.1, we can audibly inspect the synthesised results collected. Immediately we can hear a lot of noise in the synthesised results. This is due to a number of things. Firstly, the forward inference procedure is not lossless. There will inherently be a loss of resolution on the reproduction from the feature maps. Secondly, as the input to the network deals only with the magnitude portion of the spectrogram, we inherently discard the phase component. In order to estimate a best fitting phase component during re-synthesis, we make use of the well-established Griffin-Lim algorithm for estimating the output signal from the magnitude spectrogram alone [16]. Listening to the pure Griffin-Lim recreation from the original sample's magnitude spectrogram, one can already hear a level of modulation in the synthesised output stemming from the fact that the phase component is not perfect. With the loss-less nature of the inference procedure and Griffin-Lim signal estimation limitations in mind, we can listen to the synthesised output of the network's recreation of the original sample. In the output we can audibly hear the original voice speaking the original phrase, confirming the degree of recreation success.

Utilising the same Griffin-Lim synthesis approach, we are also able to synthesise audio for the random samples generated by the CD-1 and PCD trained networks. While both have a level of noise, the sample generated from the PCD trained network has characteristics much closer to that of natural speech, where the CD-1 network sample has a lot of repetition of similar sounds. This provides strong evidence in support of the claim that PCD trained networks lead to a much higher mixing rate of samples from the Markov chain, as well as a more accurately learned probability distribution from the data important for its generative capabilities [21, 45, 2]. There are a few other

sampling procedures that would be worth comparing, such as Parallel Tempering (PT) which attempts to further improve the mixing rate of the Markov chain [9].

It would be interesting to attempt to bring down the level of noise in the recreations and samples. One could attempt this by a more extensive grid-search, specifically across sparsity parameters and the number of features learned, as a lot of the noise can be attributed to multiple features being active for a given time-step. Learning more features but increasing the sparsity restriction should allow for more diverse features to be learned, less of which correlate with any given input patch. It would also be interesting to see how the samples vary as more layers are added to the model, as each layer results in a larger temporal resolution. This would provide insight into whether higher layers can learn full words from lower layers' learned phonetic features. Finally, it may be possible to re-design the network topology and adapt the training algorithm to allow for a second channel of phase data to be incorporated. If this could be successfully done a sample would consist of both portions of the spectrogram (magnitude *and* phase). One could then determine whether or not the addition of this extra data could lead to features that perform better on classification tasks, and whether or not the recreations and samples could be re-synthesised more accurately and with less noise.

6.5 Sampling Procedures: CD-1 vs PCD

In Section 2.7.3 two different sampling routines for generating synthetic samples from a RBM's learned probability distribution were discussed. Claims were presented in this section that CD-1 is often good enough for learning feature extraction, but poor at modelling the underlying distribution. PCD was argued to be better than CD-1 for modelling the underlying probability density in the data set. These insights may seem somewhat obvious from their mathematical definitions, but results collected can be investigated to verify whether or not these insights hold on a practical level.

Section 6.2 presents the optimal results achieved for both the genetic sex and speaker classification experiments. In both cases, the best performing network was the CRBM trained with CD-1 as a sampling routine. The CRBMs trained with PCD were seen to perform almost equally as well as those trained with CD-1 in the classification task. On the other hand, Section 5.4 presents synthetic samples generated by both the CD-1 and PCD trained CRBMs after 300 steps of Gibbs sampling. The results shown can only be qualitatively discussed, as experiments were not designed to quantitatively verify which generated samples bear a closer match to those from the data set's distribution. Be this as it may, it seems clear to the author that the samples generated by the PCD trained CRBM exhibit patterns that are a much closer visual match to those seen in the STFT magnitude plots of the original TIMIT data samples. Furthermore, the re-

synthesised output of these synthetic samples, as far as the author is concerned, also indicate that the PCD routine has done a better job of generating samples that closer fit the data set. This is stated as the CD-1 generated samples exhibits a very periodic high-pitched "buzz" or "humm" sound, whereas the PCD generated sample, noisy as it may be, seems to contain more audible human "babbling", similar to that described in Oord et al. [40] when the generative model was not conditioned on any given textual transcription.

The implication of these insights is that we have verified that CD-1 is good enough as a sampling routine when a CRBM is being trained to extract features for the purpose of classification. Additionally, it would seem that for the purposes of generating new synthetic samples, implementations would greatly benefit from sampling procedures that are designed to more closely approximate the RBM's true underlying distribution. This may be more concretely proven with a human believability survey, the scope of which fell outside the scope of this project due to time constraints.

6.6 Future Work

As presented throughout the course of this chapter, there are a number of additional problems that may be probed:

- Implement and train deeper CRBM layers
  - Verify feature abstractions by visualising deeper layer features and seeing if they relate to full words.
  - Verify if the deeper feature abstractions yield higher classification results.
- Fine tune CDBN through final supervised training procedure with back-propagation.
  - Verify if fine-tuning improves generative ability of the network as well as discriminative potential of the extracted features.
- Include additional channels for phase data information.
  - Verify if the additional information yields higher classification success.
  - Determine if this can improve synthesis results.
- Investigate additional MCMC sampling approaches for more accurately sampling from the RBM distribution and how this affects sample generation.
- Modify network architecture to also be conditioned on transcription and see if transcribed sentences can be synthesised as speech signal sampled from the model.
- Design and run phone classification experiment. If successful, broaden into a voice-to-text system.

- Compare results to human performance.
- Human survey to qualitatively measure effects of sampling routine on believability of synthesized sample output.

Chapter 7

Conclusion

7.1 Introduction

The aim of this project was to gain insight into Digital Signal Processing, Neural Networks, Generative Models, unsupervised training for feature extraction and Machine Learning classification tasks. The aforementioned facets were explored through the re-implementation and verification of a single CRBM layer and experiments laid out in Lee et al. [32]. This resulted in a lot of research and learnings that have been summarised and outlined in Chapter 2, which will have brought the reader up to speed with the background theory needed to understand the implementation and experiments. In addition to purely theoretical learnings, the project aimed to practically implement the feature extraction architecture and experiments from first principles to provide practical learnings, the details of which were outlined in Chapters 3 and 4. Results were collected from the implementation and experiments and have been presented and discussed in Chapters 5 and 6. The remainder of this conclusion reviews the results and learnings gathered during the course of the project and places them in context among the related work presented in Chapter 2.

7.2 Feature Extraction & Audio Data

Machine Learning classification tasks involve accurately mapping some observations, or features, of a data sample to a desired target label or class that the data sample fits into. The ability of the data observations to be easily mapped to the correct class label will rely on using a compact and decorrelated feature set representing meaningful information within the observations [26, 3]. This makes it clear that coming up with a good feature set is a crucial step in the whole classification pipeline. When it comes to audio data it is not immediately apparent what useful features could be extracted from the raw data. Fortunately much work has been done on the subject.

7.2.1 Manual Feature Extraction

Humans tend to bias information in data such as audio signals that can be perceived with the ears [33, 47]. These human-perceptually relevant features, which are often enabled through Fourier analysis, have taken many years to define and craft. They have been well studied and implemented by a number of voice-to-text and Music Information Retrieval (MIR) systems [49, 13, 33, 48, 34]. The draw-backs to manually crafting your feature set are the complexity of the mathematics involved, and the potential to throw away information that may prove useful that is simply not perceivable to human ears and understanding.

7.2.2 Automatic Feature Extraction

It is due to the complexity of information within audio data that automatic approaches to its discovery and extraction are being explored.

Raw Audio As Features

One approach to automated feature extraction is to simply use the raw data as the features. This is obviously a very simplistic approach, but has shown some success in the time-domain as well as frequency-domain representations [40, 10]. The issues that arise from this, however, are potentially poor results and dealing with the high dimensionality of the raw data. In order to usefully analyse more than a few milliseconds of audio, the authors in Oord et al. [40] develop a novel and complex convolutional operation that shrinks the representation over time, allowing for a reduced dimensionality to be considered for larger time spans. This operation is much more computationally complex than those explored in this dissertation, but has produced some exciting results.

Autoencoders

Autoencoders are neural networks that are trained to recreate their inputs. They have been shown to be successful as feature extractors by encoding more compact representations of their inputs at their middle layers, from which the original information can be retrieved [23]. They have also been applied to audio data in order to create binary codings of speech spectrograms [8]. This promising work involves initially training a RBM greedily, layer by layer in a routine outlined in Bengio [1]. Once the desired architecture has had its layers trained, the network is converted into a deep autoencoder and fine-tuned through a supervised learning approach. The RBM style pre-training has shown to greatly improve autoencoder training, and more generally neural network convergence [23, 8]. The final step of network conversion and fine-tuning turns the generative RBM model into a discriminative network only, losing the ability to synthesize

new samples from the model. Both of these points motivate further study and development of RBM models as applied to audio data for their discriminative (classification) and generative properties.

Restricted Boltzmann Machines

A number of previous works have explored the use of RBMs to perform automated feature extraction, many operating on audio data, with varying amounts of success [9, 20, 25, 30, 31, 32, 36]. The networks result in the automatic learning of features that can be used to describe the underlying probability density of their training set. The ability for these networks to be trained layer by layer has the benefit that adding additional layers does not require retraining of lower layers, and ensures that deeper layers are being fed fully trained lower level features during their training, resulting in a better hierarchical feature representation at higher and higher levels of abstraction [1, 31]. The convolutional version of these networks, as per Lee et al. [32], was chosen as the technique to investigate for this project as there has been much success with convolutional approaches to image recognition, as well as displaying the highest level of success in using the learned features to perform a useful audio classification task.

7.3 CRBM Feature Verification Results

Experimental results presented and discussed in Chapters 5 and 6 confirm that features automatically learned and extracted from CRBM networks are clearly useful for complex classifications from high dimensional data such as audio. This result concurs with Lee et al. [32], on which the implementation and experiments were based. This allows the interested scientist to focus less on becoming an expert in feature crafting techniques that suit the type of data at hand and focus more on experimenting with network topologies and machine learning algorithms. Only a single layered network was considered for this project due to time and computing power constraints, yet results produced still show how useful its learned feature extraction can be in both a genetic sex classification task, as well as a 168-way speaker identification task.

7.4 RBMs As Statistical Models

It has been discussed that data sets such as the TIMIT data set describe a probability distribution that can be (approximately) automatically learned. Restricted Boltzmann Machines behave as statistical models in this way, and can encode this probability distribution with regards to some hidden latent variable. Given that the RBM has learned an accurate approximation of the training data's distribution, it can now be sampled from to synthesize previously unseen data points that should fit into the distribution.

As the training function emphasizes the difference between any training sample and a sample from the model's current true distribution, it is clear that choice of sampling technique can be crucial to how good the approximation of the final model is. Hinton [21] indicates that simple sampling techniques such as CD-1, discussed in Section 2.7.3, perform well enough for feature extraction for classification tasks. The results presented in Chapter 5 confirm this. Models trained with sampling procedures that better approximate the RBM distribution were not seen to perform better than those trained with CD-1. It is qualitatively discussed, however, that models trained with more accurate sampling strategies are able to generate samples that are much closer to the training set's distribution. This implies that if generating good samples is the desired use-case, that more effective sampling procedures should be employed during training.

7.5 Final Thoughts

While initial results are promising, there is still a lot of work to be done to improve classification accuracies and synthesis results, including adaptations to include phase spectrogram data, convolution filter methods to grow the temporal resolution of the model over time and comparing MCMC sampling techniques and their ability to approximate the model's learned distribution. In addition, research avenues for signal recreation and speech generation using the CRBM architecture have been identified and discussed in Chapter 6. It is hoped that this project inspires additional work and focus on these topics.

Bibliography

[1] Yoshua Bengio. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[2] Mathias Berglund and Tapani Raiko. Stochastic gradient estimate variance in contrastive divergence and persistent contrastive divergence. *arXiv preprint arXiv:1312.6002*, 2013.

[3] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[4] Léon Bottou. Online learning and stochastic approximations. *On-line learning in neural networks*, 17(9):142, 1998.

[5] Pierre Brémaud. *Markov chains: Gibbs fields, Monte Carlo simulation, and queues*, volume 31. Springer Science & Business Media, 2013.

[6] Emmanuel J Candès and Michael B Wakin. An introduction to compressive sampling. *IEEE signal processing magazine*, 25(2):21–30, 2008.

[7] Augustin Cauchy. Méthode générale pour la résolution des systemes déquations simultanées. *Comp. Rend. Sci. Paris*, 25(1847):536–538, 1847.

[8] Li Deng, Michael L Seltzer, Dong Yu, Alex Acero, Abdel-rahman Mohamed, and Geoffrey E Hinton. Binary coding of speech spectrograms using a deep auto-encoder. In *Interspeech*, pages 1692–1695. Citeseer, 2010.

[9] Asja Fischer and Christian Igel. Training restricted boltzmann machines: An introduction. *Pattern Recognition*, 47(1):25–39, 2014.

[10] James L Flanagan. *Speech analysis synthesis and perception*, volume 3. Springer Science & Business Media, 2013.

[11] Imola K Fodor. A survey of dimension reduction techniques. Technical report, Lawrence Livermore National Lab., CA (US), 2002.

[12] Gerald B Folland. *Fourier analysis and its applications*, volume 4. American Mathematical Soc., 1992.

[13] Jonathan T Foote. Content-based retrieval of music and audio. In *Voice, Video, and Data Communications*, pages 138–147. International Society for Optics and Photonics, 1997.

[14] 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*, 10(5):0, 1993.

[15] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

[16] Daniel Griffin and Jae Lim. Signal estimation from modified short-time fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32(2): 236–243, 1984.

[17] Jacques Hadamard. *Mémoire sur le problème d'analyse relatif à l'équilibre des plaques élastiques encastrées*, volume 33. Imprimerie nationale, 1908.

[18] Fredric J Harris. On the use of windows for harmonic analysis with the discrete fourier transform. *Proceedings of the IEEE*, 66(1):51–83, 1978.

[19] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[20] G. E. Hinton. Deep belief networks. *Scholarpedia*, 4(5):5947, 2009. doi: 10.4249/scholarpedia.5947. revision #91189.

[21] Geoffrey Hinton. A practical guide to training restricted boltzmann machines. *Momentum*, 9(1):926, 2010.

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

[23] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[24] A James Hudspeth. How the ear's works work. *Nature*, 341(6241):397, 1989.

[25] Navdeep Jaitly and Geoffrey Hinton. Learning a better representation of speech soundwaves using restricted boltzmann machines. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 5884–5887. IEEE, 2011.

[26] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160:3–24, 2007.

[27] Hugo Larochelle. Neural networks [5.2] : Restricted boltzmann machine - inference. [YouTube video], Nov. 15 2013. URL https://youtu.be/lekCh_i32iE?t=4m38s. Accessed May 24, 2017.

[28] Yann LeCun, Corinna Cortes, and Christopher JC Burges. Mnist handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2, 2010.

[29] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521 (7553):436–444, 2015.

[30] Honglak Lee, Chaitanya Ekanadham, and Andrew Y Ng. Sparse deep belief net model for visual area v2. In *Advances in neural information processing systems*, pages 873–880, 2008.

[31] 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*, pages 609–616. ACM, 2009.

[32] Honglak Lee, Peter Pham, Yan Largman, and Andrew Y Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In *Advances in neural information processing systems*, pages 1096–1104, 2009.

[33] Thomas Lidy, Georg Pölzlbauer, and Andreas Rauber. Sound re-synthesis from rhythm pattern features - audible insight into a music feature extraction process. *Proceedings of the International Computer Music Conference (ICMC)*, pages 93–96, 2005.

[34] Beth Logan et al. Mel frequency cepstral coefficients for music modeling. In *ISMIR*, 2000.

[35] Brian McFee, Matt McVicar, Oriol Nieto, Stefan Balke, Carl Thome, Dawen Liang, Eric Battenberg, Josh Moore, Rachel Bittner, Ryuichi Yamamoto, et al. librosa 0.5.1, 2017.

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

[37] Moxfyre. Aliasing, 2017. URL https://upload.wikimedia.org/wikipedia/commons/2/28/Alia [Online; accessed September 7, 2017].

[38] NumPy. Numpy. *NumPy Numpy. Scipy Developers*, 2013.

[39] CUDA Nvidia. Programming guide, 2010.

[40] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[41] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[42] Claude Elwood Shannon. Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21, 1949.

[43] Kihyuk Sohn, Dae Yon Jung, Honglak Lee, and Alfred O Hero. Efficient learning of sparse, distributed, convolutional feature representations for object recognition. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2643–2650. IEEE, 2011.

[44] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016. URL `http://arxiv.org/abs/1605.02688`.

[45] Tijmen Tieleman. Training restricted boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th international conference on Machine learning*, pages 1064–1071. ACM, 2008.

[46] tutorialspoint.com. Artificial intelligence - neural networks, 2017.

[47] Rainer Typke, Remco C Veltkamp, and Frans Wiering. Searching notated polyphonic music using transportation distances. In *Proceedings of the 12th annual ACM international conference on Multimedia*, pages 128–135. ACM, 2004.

[48] George Tzanetakis and Perry Cook. Musical genre classification of audio signals. *Speech and Audio Processing, IEEE transactions on*, 10(5):293–302, 2002.

[49] E. Wold, T. Blum, D. Keislar, and J. Wheaton. Content-based classification, search, and retrieval of audio. *MultiMedia, IEEE*, 3(3):27–36, 1996.

Appendix A

First Appendix

A.1 Audio Files

1. TIMIT Audio Sample:
   https://drive.google.com/open?id=1JJcMfRbMFOo2ZzA9b17uWAwF4L07HkgW
2. TIMIT Audio Sample Recreated From Magnitude Spectrogram Using Griffin-Lim:
   https://drive.google.com/open?id=1-cwJxlpcGrndeVe-WouzzFkqnSIySa10
3. CD-1 TIMIT Audio Network Recreation Synthesis:
   https://drive.google.com/open?id=1qUFyDfUcXG7kPXOLmChKriJGqIDyrKoQ
4. CD-1 300-Step Gibbs Sample Synthesis From Random Start:
   https://drive.google.com/open?id=13KZsL3AN5fpOzMCjy1ZyQaPI3JuAVUQH
5. PCD 300-Step Gibbs Sample Synthesis From Random Start:
   https://drive.google.com/open?id=1P9OTtsvCy93LFCPmcf8O3DpslGnQ9hqC