Improving inter-speaker performance in unsupervised phone discovery using speaker identity side information

ARVID FAHLSTRÖM MYRMAN

Master in Computer Science
Date: March 2, 2017
Supervisor: Giampiero Salvi
Examiner: Olov Engwall
Swedish title: Detta är den svenska översättningen av titeln
School of Computer Science and Communication

Abstract

English abstract goes here. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Sammanfattning

Träutensilierna i ett tryckeri äro ingalunda en faktor där trevnadens ordningens och ekonomiens upprätthållande, och dock är det icke sällan som sorgliga erfarenheter göras ordningens och ekon och miens därmed upprätthållande. Träutensilierna i ett tryckeri äro ingalunda en oviktig faktor, för trevnadens ordningens och och dock är det icke sällan.

Contents

Chapter 1

Introduction

Background

Automatic speech recognition (ASR) is generally framed as a supervised task, where both audio data and the corresponding transcription is available, and the problem is to develop a model that can mimic this mapping from speech to text. However, developing such data is expensive, both in terms of time and money, as it involves painstakingly transcribing many hours of speech and aligning the transcription in time by hand. As a result, there is a notable lack of high-quality data for speech recognition for a majority of languages around the world. An important question is thus whether it is possible to make use of untranscribed, or unlabelled, data to develop ASR for such low-resource languages. Unsupervised learning in this manner may also provide insight into the linguistic structure of languages, or the language acquisition of infants.

Unsupervised speech recognition is an area of active research. One source of research in this area is the Zero Resource Speech Challenge (Versteegh et al. 2015), which was developed with the goal of finding linguistic units (track 1) or longer recurring word-like fragments (track 2) in speech. Models are to be trained using only speech data, voice activity information, and speaker identity information. In the spirit of the Zero Resource Speech Challenge, this thesis will follow the first track of the challenge, using the same training data and evaluation procedure.

One approach that has proved itself successful in modelling linguistic units is to first discover recurring speech fragments, and then use these fragments as constraints to construct features where speech frames corresponding to the same sound are similar (Synnaeve et al. 2014; Thiolliere et al. 2015). One motivation for taking this top-down approach is that sounds that in reality correspond to the same linguistic unit may seem very different when inspecting speech at specific time instances, especially when comparing different speakers; however, when viewed at a larger time scale, patterns from e.g. recurring words are easier to find (Jansen et al. 2013).

Simpler approaches such as direct clustering of unlabelled speech has also been shown to perform well (Chen et al. 2015). This work seeks to find whether it is possible to combine the two approaches, by first inferring a probabilistic model from unlabelled speech, and afterwards improving on this model using speech fragment information. This approach is similar to the one of Jansen et al. (2013), where a universal background model in the form of a Gaussian mixture model is inferred and later partitioned, with the difference that the partitioning is done approximately using a linear siamese model inspired by Synnaeve et al. (2014), taking advantage of both same-class and different-class fragment information.

Objective

This work seeks to find speaker-invariant speech features which can be used to discriminate between different linguistic units in a robust manner. We focus in particular on simpler, interpretable models that improve on features inferred from unlabelled data, thus also making use of the full set of unlabelled data, in addition to speech fragment information. The goal is for the model to noticably improve on the input features, as well as to achieve competitive results in the context of the first track of the Zero Resource Speech Challenge.

Chapter 2

Theory

This chapter provides an overview of topics that serve as a base for the rest of the thesis. Topics covered include speech recognition, basic concepts in machine learning, and artificial neural networks.

Audio processing for speech recognition

An important step in most speech recognition applications is feature extraction: From a raw audio signal we wish to extract information (features) that can be used to effectively process or model the speech to achieve some desired result. As the exact types of features used vary depending on the model used and the goal of the application, this section will focus on some particularly common features used in speech recognition. Additionally, some standard methods of processing speech that take into consideration the sequential and dynamic nature of speech will be discussed.

This section serves as a short introduction to signal processing for speech recognition. For a more in-depth description, see a book on signal processing such as Quatieri (2002), or see Huang et al. (2001) for an introduction to speech recognition in particular.

Audio signals

In the real world speech takes the form of a pressure wave generated as air is pushed through the vocal tract. The pressure wave as perceived from a single point in space can be described as a continuous signal $x(t)$ $(t \in \mathbb{R})$ that varies smoothly in time, with $x(t)$ at each time t describing the amplitude of the wave relative to the ambient pressure. Our perception of the signal depends not on the absolute value of $x(t)$ at any given time instant, but rather on how it varies in time. As an example, take a simple sine wave $x(t) = \sin(2\pi f t)$; though the signal oscillates continuously in time, a human would perceive the signal as a single constant sound of frequency f.

notation for domain?

However, while the physical signal is continuous, digital computers are unable to handle signals with an infinitely high time resolution, and as a consequence the signal must be somehow discretised before it can be processed further by speech applications. This is done by taking samples of the signal at fixed time steps given by a sampling frequency f_s (e.g. 16 000 Hz), specifying the number of samples taken per second. The resulting sampled signal $x[n]$ $(n \in \mathbb{Z})$ is a discrete-time approximation of the original signal $x(t)$.

Short-time Fourier transform

maybe mention window functions?

unit of energy?

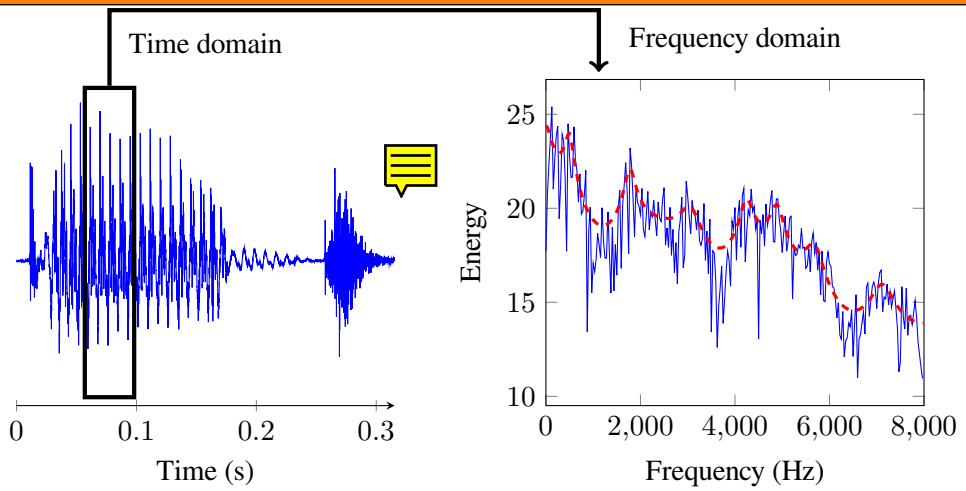

Figure 2.1: Example of running a short-time Fourier transform on a 25 ms section of a recording of the word “bed”. The dashed line shows the “envelope”, or overall shape, of the energy spectrum. The peaks and valleys of the spectrum are characteristic of sonorant sounds such as vowels.

something about the Nyquist frequency

To mirror how humans perceive audio signals it is useful to transfer the signal to some form that better captures the fluctuations of the signal. One way is to analyse the *frequency content* of the signal using the Fourier transform. The Fourier transform approximates the signal using a sum of sine and cosine waves of different frequencies, giving the amplitude of each such wave, which in turn can be interpreted as the energy content of the signal at the corresponding frequency. In particular, the discrete Fourier transform (DFT), defined as

$$X(k) = \sum_{n=0}^{N-1} x[n] e^{-j2\pi kn/N} \tag{2.1}$$

where j is the imaginary unit and $k \in [0, N-1]$ corresponds to the frequency $f = \frac{k}{N} f_s$, gives the frequency content of a finite discrete-time signal of length N samples. The energy density, or the energy distribution over frequency, is given by $S(k) = |X(k)|^2$. The DFT can be calculated in $O(N \log N)$ asymptotic time using the fast Fourier transform (FFT) algorithm, especially in the case where N is chosen to be a power of 2 (Cooley and Tukey 1965).

sources for the paragraph below

The DFT makes certain assumptions regarding the nature of the signal. In particular, it assumes that the signal is periodic, which is emphatically not true in general for speech signals. However, a speech signal can be thought to be *approximately periodic* over a very short time period. This gives rise to the so-called short-time Fourier transform (STFT), where the DFT is calculated repeatedly on short sections of the signal using a sliding window; see figure 2.1 for an example. A typical window length is 25 ms, and it is generally shifted forward about 10 ms between each DFT calculation. The result of the STFT is the 2D Fourier transform $X(m, k)$ over time step m and frequency k, with corresponding energy density $S(m, k) = |X(k, m)|^2$.

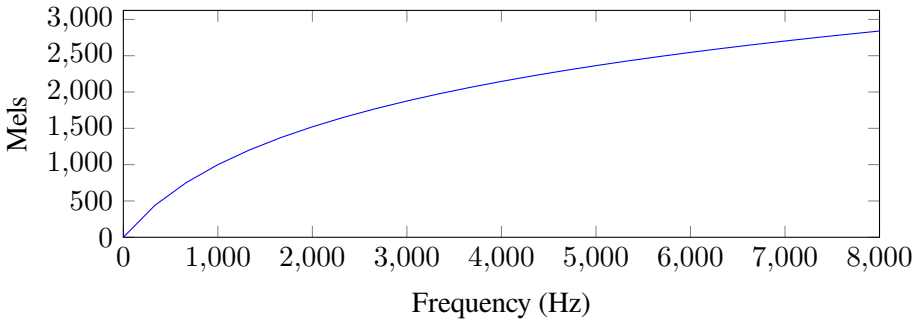

Figure 2.2: The mel scale as it corresponds to the standard frequency scale. Higher frequencies are closer together on the mel scale than lower frequencies.

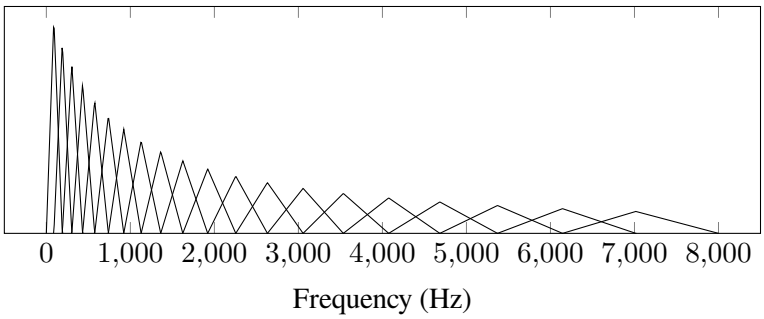

Figure 2.3: 20 triangular filters spaced linearly along the mel scale from 0 Hz to 8000 Hz. The peaks of the filters are scaled as to ensure constant area.

Mel-scale filter banks

There are several problems with working directly with the output of the STFT. One is that the output is very high-dimensional, as N is often chosen to be in the 512 to 2048 range for the DFT. Unless very large amounts of data is available, this causes data sparsity issues, which can cause many kinds of models to underperform. Additionally, we are not really interested in the exact energy at every frequency step, but would rather like to extract the overall shape of the spectrum, preferably in a way that also suppresses some of the noise.

Finally, all frequencies are not created equal, as the human ear does not discriminate between higher frequencies to the same extent as between lower frequencies. To model this phenomenon, scales in which a change in pitch corresponds roughly linearly to the subjective change in pitch perceived by a human have been empirically developed. One such scale, ubiquitous in speech technology, is the mel scale, which was developed through experiments where participants were told to produce a tone with half the perceived pitch of a reference tone (Stevens et al. 1937). A frequency f can be converted to the mel scale through the following relation:

$$\text{mel}(f) = 1127 \log\left(1 + \frac{f}{700}\right) \tag{2.2}$$

where log is the natural logarithm. As can be seen in figure 2.2, as f grows larger, the difference $\text{mel}(f+\varepsilon) - \text{mel}(f)$ becomes smaller.

Addressing both the issues of dimensionality and perception, we construct the mel-scale filter bank. The filter bank is a set of L triangular filters, with the middle points of the filters spaced linearly on the mel scale. In other words, if $f_1, f_2, \ldots, f_L$ are the middle points of the filters specified in Hz,

$\text{mel}(f_k) - \text{mel}(f_{k-1}) = \text{mel}(f_{k+1}) - \text{mel}(f_k), k \in [2, L-1]$. The start and end points of the filters are the middle points of the previous and following filters, respectively, with the exception of the first filter whose start point is specified by a lower bound f_{low}, and the last filter whose end point is given by a higher bound f_{high}. The height of the peak of each filter can vary; common approaches are to ensure that the filters have either constant height or constant area. An illustration of a filter bank is given in figure 2.3. Each filter is applied to the energy spectrum of the audio signal, giving the filter bank output $E(m,l) = \sum_{k=0}^{N-1} V_l(k)S(m,k)$, where $V_l(k)$ is the value of the $(l+1)$th mel-scale filter at frequency k. The resulting L values at each time step m form a rough approximation of the energy spectrum, with the resolution being higher for low frequencies than for high frequencies. L is commonly chosen to be in the 20 to 40 range, significantly lowering the dimensionality of the data.

Mel frequency cepstral coefficients

In certain applications, it is beneficial if the features generated are relatively decorrelated. For instance, this is the case when modelling the features using multivariate Gaussian distributions, where if the features are decorrelated the covariance matrix can be approximated using a diagonal matrix, significantly reducing the number of parameters that need to be learnt.

Unfortunately, the mel-scale filter bank outputs are *not* decorrelated, as neighbouring frequencies tend to take on similar values in the energy spectrum. However, they can be made roughly decorrelated by taking the discrete cosine transform (DCT) of the logarithm of the filter bank outputs, given as

$$C(m,c) = \sum_{l=0}^{L-1} \log(E(m,l)) \cos\left[\frac{2\pi}{L}\left(l+\frac{1}{2}\right)c\right] \tag{2.3}$$

for $c \in [0, L-1]$. The DCT approximates the input using cosine waves in a similar manner to how the Fourier transform works, with each coefficient c indicating how similar the input is to the cosine wave with frequency c/L. true?

The resulting values are known as the mel-frequency cepstral coefficients (MFCCs). Usually only the first 13 or so coefficients are used at each time step, i.e. the coefficients corresponding to $c \in [0, 12]$, once again reducing the dimensionality of the data.

Modelling evolution over time

Speech is inherently sequential. This means that our perception of speech is not dependent on particular absolute values of the speech signal at particular points in time, but rather on how the signal evolves over time, and the exact realisation of individual speech sounds is formed through complex interplay with neighbouring sounds. In addition, many speech sounds change over time by nature; examples include the diphthong /aɪ/ in the word *my* /maɪ/, or the affricate /tʃ/ in *teach* /tiːtʃ/, both acting as single units of speech, despite the onset and offset of the sounds being significantly different acoustically.

Thus, it is a difficult task to recognise a speech sound based only on a single frame of audio. Instead, the model needs some way of incorporating information about the context of the frame. This can be done both at the model level and the feature level. Model-based approaches include hidden Markov models (HMMs), which encode information about how the speech can change in time in the form of probabilities, and recurrent neural networks (RNNs), which can take sequences as input and automatically find temporal patterns.

Feature-based approaches, instead, incorporate temporal information directly into the features. One simple way is to extend each speech frame to include not only the current frame, but also the neighbouring frames before feeding it to the model. For instance, if our features consisted of the outputs of a mel-scaled filter bank of size 40 at different time steps m and we wanted to include a context

of 2 frames in both directions it time, the frame at each m would be extended to include the frames at $m-2$, $m-1$, m, $m+1$ and $m+2$, resulting in a feature vector of size $40 \cdot 5 = 200$ at each time step.

better source for below

Another feature-based approach is to include approximations of the temporal derivatives in the feature vector, most commonly the first-order (velocity) and second-order (accerelation) derivatives. We do this by approximating the feature vector sequence using a second-order polynomial $f(k) = a + bk + ck^2$ and taking its derivative $f'(k) = b + 2ck$. Let $y_{-n}, \ldots, y_{-1}, y_0, y_1, \ldots, y_n$ be a sequence of feature values (e.g. the values corresponding to a single MFCC), where we are interested in the temporal derivative at the point corresponding to y_0, i.e. at $k = 0$. n is the number of points at each side of y_0 that we want to use to estimate the polynomial. We wish to find the coefficients that minimise

$$\sum_{k=-n}^{n} (f(k) - y_k)^2. \tag{2.4}$$

As the derivative of $f(k)$ at $k = 0$ is $f'(0) = b$, we only need to find a solution for b. We minimise the error function by taking the gradient with respect to b and setting it to 0:

$$\frac{\partial}{\partial b} \sum_{k=-n}^{n} (f(k) - y_k)^2 = 0 \tag{2.5}$$

$$\frac{\partial}{\partial b} \sum_{k=-n}^{n} (a + bk + ck^2 - y_k)^2 = 0 \tag{2.6}$$

$$\sum_{k=-n}^{n} 2k(a + bk + ck^2 - y_k) = 0 \tag{2.7}$$

$$\sum_{k=-n}^{n} ak + \sum_{k=-n}^{n} bk^2 + \sum_{k=-n}^{n} ck^3 = \sum_{k=-n}^{n} ky_k. \tag{2.8}$$

By antisymmetry we see that $\sum_{k=-n}^{n} ak = \sum_{k=-n}^{n} ck^3 = 0$, leaving us with

$$\sum_{k=-n}^{n} bk^2 = \sum_{k=-n}^{n} ky_k \tag{2.9}$$

$$b = \frac{\sum_{k=-n}^{n} ky_k}{\sum_{k=-n}^{n} k^2} \tag{2.10}$$

$$f'(0) = \frac{\sum_{k=1}^{n} k(y_k - y_{-k})}{2\sum_{k=1}^{n} k^2}. \tag{2.11}$$

Thus, in general, to approximate the first-order temporal derivative at a point in time t, also known as the *delta* value at t, we have

$$\Delta y_t \approx \frac{\sum_{k=1}^{n} k(y_{t+k} - y_{t-k})}{2\sum_{k=1}^{n} k^2} \tag{2.12}$$

which is the formula used by toolkits such as HTK (Young et al. 2005). The second-order derivative, or the *delta-delta* values, can be obtained by repeating the process using the delta values.

Dynamic time warping

Rather than training a model to perform speech recognition, it is sometimes of interest to directly measure the similarity of two utterances. However, the dynamic nature of speech makes this difficult: A

single speaker if asked to repeat a word two times will not pronounce the word exactly the same both times, and the length of the utterances will also differ slightly. Dynamic time warping (DTW) attempts to deal with this by finding the best alignment of the frames of the two utterances and measure the similarity based on this alignment.

Let $d(\mathbf{x}, \mathbf{y})$ be some local measure of the distance between the feature vectors $\mathbf{x}$ and $\mathbf{y}$ (e.g. the Euclidean distance) so that $d(\mathbf{x}, \mathbf{y})$ is larger the further apart the feature vectors are. Let $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ and $\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_m\}$ be sequences of feature vectors, and let $\mathbf{X}_{k:l}$ denote the subsequence of $\mathbf{X}$ starting at k and ending at l. We wish to find a global distance measure $D(\mathbf{X}, \mathbf{Y})$ as a sum of the local distances between the feature vectors, based on some alignment that minimises this distance. All elements in both sequences must be used in the final alignment, but elements may be repeated at will to serve as padding.

This can be expressed through the recurrence

$$\begin{aligned} D(\mathbf{X}_{1:k}, \mathbf{Y}_{1:l}) = d(\mathbf{x}_k, \mathbf{y}_l) + \min(&D(\mathbf{X}_{1:k-1}, \mathbf{Y}_{1:l-1}), \\ &D(\mathbf{X}_{1:k-1}, \mathbf{Y}_{1:l}), \\ &D(\mathbf{X}_{1:k}, \mathbf{Y}_{1:l-1})) \end{aligned} \tag{2.13}$$

with the base cases

$$D(\mathbf{X}_{1:0}, \mathbf{Y}_{1:0}) = 0 \tag{2.14}$$

$$D(\mathbf{X}_{1:0}, \mathbf{Y}_{1:k}) = D(\mathbf{X}_{1:k}, \mathbf{Y}_{1:0}) = \infty. \tag{2.15}$$

Padding with ∞ is required to ensure that the alignment starts with the first element in both sequences.

The way the recurrence is defined enables evaluation of $D(\mathbf{X}, \mathbf{Y})$ in $O(nm)$ time using dynamic programming. By saving backpointers during the calculation, it is also possible to retrieve the actual alignment.

Machine learning

Machine learning can roughly be described as the practice of automatically finding patterns in data, and using these patterns to make future predictions or perform decision making. The learning process generally takes the form of setting up an appropriate mathematical or statistical model, and automatically changing the parameters of the model to fit the data. Machine learning is commonly employed in a variety of fields such as speech recognition, computer vision and natural language processing (NLP).

Using text parsing in NLP as an example, machine learning provides several advantages over the classic approach of hand-engineered rules written by human experts, including:

make better arguments

- Grammatical rules are inferred based on actual data, rather than the expert's conception of how the language works.
- A computer can quickly go through an amount of data that would be far too vast for a human expert to analyse by hand.
- A machine learning model can incorporate statistical information learnt from large amounts of data, enabling it to return multiple possible interpretations along with confidence scores.

Similar advantages can be seen in other fields, such as speech recognition where it is simply not feasible to construct hand-written rules that can identify speech sounds from raw audio data.

This section will lightly touch upon machine learning concepts relevant to this thesis; for a proper introduction to the field, see Murphy (2012).

Important concepts

Supervised and unsupervised learning

A large number of techniques in machine learning can be broadly considered to be either supervised techniques, which take a set of data along with corresponding labels and try to learn the mapping from the data to the labels, or unsupervised techniques, which try to find "interesting" (as defined by the task at hand) patterns in unlabelled data.

As an example, consider the task of speech recognition. The data set used to train our model consists of speech data along with a set of phonetic transcriptions, so that what is being said at each time instant is known. Our task is to try to learn this mapping from speech to transcription, taking advantage of all available data. This is a typical example of supervised learning, as we have a known "ground truth" that we are trying to replicate.

On the other hand, consider the case where our speech data is unlabelled, so that we do not know what is being said in a given utterance. Without the ground truth we do not have a reference we can use to learn the mapping from speech signal to transcription. Instead, our task is reduced to trying to find patterns in the data, by for example identifying repeating segments in the speech that could possibly correspond to speech sounds, or even whole words. Note, however, that even if we are able to correctly identify words in the speech, we still do not know the corresponding orthographic transcription. This is an example of unsupervised learning.

Regression

Regression is a supervised task where we are given input data $\mathbf{x} \in \mathbb{R}^n$—here real, though discrete input data is also common—and corresponding continuous output data $\mathbf{y} \in \mathbb{R}^m$, and our task is to find a function $f : \mathbb{R}^n \to \mathbb{R}^m$ that best preserves the relationship between input and output, and that can be used to predict output values for future input data.

A typical example might be predicting house prices, where the input data consists of information such as floor area, garden area, proximity to public transport, etc., and the output is a scalar indicating the price.

Classification

Classification is a special case of regression where the output is a discrete class. Thus, the problem is finding a function $f : \mathbb{R}^n \to C$ where C is the set of possible classes. In some cases a data point may belong to more than one class at a time, in which case the mapping function can be defined as $f : \mathbb{R}^n \to \{0, 1\}^c$, where c is the number of possible classes, and $f(\mathbf{x})_k$ is 1 if $\mathbf{x}$ belongs to class k; this is known as multi-label classification.

A common classification task is image classification, where the objective is to identify the object or objects present in an image. The input is the value of each pixel in the image, and the output is the class or set of classes corresponding to the object(s) in the image.

Clustering

Clustering is an unsupervised task, where the goal is to somehow group the input data into distinct classes, such that data points in one class are more similar to each other than to data points in other classes. Both the concept of similarity and the interpretation of the different classes depends on the problem at hand. One example of clustering is the grouping of speech frames generated from an audio signal into distinct phonetic classes, with no prior knowledge of what phonetic classes are available.

Embedding

In some cases we do not have access to the true class corresponding to a data point, but we *do* have information such as whether two given data points belong to the same class or not, or what context the data points appear in. Using this information we wish to project the data onto a new space where similar data points are close, while dissimilar points are distant. This projection is referred to as an embedding, and the techinque has seen use in areas such as face recognition, where faces are projected onto a space where similarity can be measured more easily, and natural language processing, where words, sentences or even whole documents are converted into real-valued vectors that capture some semantic information.

K-means clustering

A simple approach to clustering is known as K-means clustering. Here, K cluster centers are initialised, often randomly, and then iteratively updated to better reflect the data. At each iteration every data point is assigned to the cluster whose center is closest to it in terms of the Euclidean distance, and the center of each cluster is then set to the mean of the data points assigned to the cluster. Once the cluster centers converge (i.e. stop updating), the training stops.

Gaussian mixture models

While K-means clustering is simple to implement, it also makes some hidden assumptions, such as each cluster being spherical, which make it a bad fit for many types of data. A better assumption in many cases is that each data point was generated by one of K multivariate Gaussian distributions, each Gaussian k having mean $\boldsymbol{\mu}_k$ and covariance matrix $\boldsymbol{\Sigma}_k$. We denote the probability of the kth Gaussian generating a data point as $p(k \mid \boldsymbol{\theta}) = \pi_k$, where $\sum_{k=1}^{K} \pi_k = 1$. The probability of seeing a data point $\mathbf{x}$ is then described by

$$p(\mathbf{x} \mid \boldsymbol{\theta}) = \sum_{k=1}^{K} p(k \mid \boldsymbol{\theta}) p(\mathbf{x} \mid k, \boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{2.16}$$

where $\boldsymbol{\theta}$ is the parameters of the model:

$$\boldsymbol{\theta} = \{\pi_1, \pi_2, \ldots, \pi_K, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_K, \boldsymbol{\Sigma}_1, \boldsymbol{\Sigma}_2, \ldots, \boldsymbol{\Sigma}_K\}. \tag{2.17}$$

This is known as a (finite, as the number of components K is set *a priori*) Gaussian mixture model (GMM). Clustering using a GMM is performed by initialising the parameters $\boldsymbol{\theta}$ to some (e.g. random) value, and then iteratively updating the parameters using the expectation maximisation (EM) algorithm to maximise the probability of the model having generated the data. See Murphy (2012) for a detailed description of EM for GMMs.

After training, the posterior distribution $p(k \mid \mathbf{x}, \boldsymbol{\theta})$ can be calculated as

$$p(k \mid \mathbf{x}, \boldsymbol{\theta}) = \frac{p(k \mid \boldsymbol{\theta})p(\mathbf{x} \mid k, \boldsymbol{\theta})}{p(\mathbf{x} \mid \boldsymbol{\theta})} = \frac{p(k \mid \boldsymbol{\theta})p(\mathbf{x} \mid k, \boldsymbol{\theta})}{\sum_{l=1}^{K} p(l \mid \boldsymbol{\theta})p(\mathbf{x} \mid l, \boldsymbol{\theta})} = \frac{\pi_k \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{l=1}^{K} \pi_l \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)}, \tag{2.18}$$

giving the probability of $\mathbf{x}$ belonging to class k.

Artificial neural networks

Artificial neural networks (ANNs) are a family of machine learning models loosely inspired by biological neural networks. Though different types of ANNs function quite differently from each other, a common theme is that they are composed of a network of units, or neurons, each performing a relatively simple task. The power of the model comes from combining a large amount of units to form a single, complex model.

This section is mainly concerned with a specific type of neural network, namely the feedforward neural network. The feedforward neural network is a regression function $f : X \to Y$ that takes an n-dimensional input $\mathbf{x} \in X, X \subseteq \mathbb{R}^n$ and returns an m-dimensional output $\mathbf{y} \in Y, Y \subseteq \mathbb{R}^m$. Though the model is inherently a regressor, representing both input and output as real values, feedforward neural networks have been successfully applied to e.g. classification by interpreting the output $\mathbf{y}$ as a probability distribution, defining the probability of $\mathbf{x}$ belonging to class k as $P(k \mid \mathbf{x}) = y_k$.

For a recent detailed text on ANNs in the context of deep learning, see Goodfellow et al. (2016).

Linear models

Consider the problem of predicting m scalar output variables $y_1, y_2, \ldots, y_m$ using a weighted linear combination of n input variables $x_1, x_2, \ldots, x_n$: $y_j = \sum_{i=1}^{n} x_i w_{ij} + b_j$. In matrix notation we write this as

$$\mathbf{y} = \mathbf{x}\mathbf{W} + \mathbf{b} \tag{2.19}$$

where

$$\mathbf{x} = \begin{pmatrix} x_1 & x_2 & \cdots & x_n \end{pmatrix} \tag{2.20}$$

$$\mathbf{W} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & \cdots & w_{nm} \end{pmatrix} \tag{2.21}$$

$$\mathbf{b} = \begin{pmatrix} b_1 & b_2 & \cdots & b_m \end{pmatrix} \tag{2.22}$$

$$\mathbf{y} = \begin{pmatrix} y_1 & y_2 & \cdots & y_m \end{pmatrix}. \tag{2.23}$$

$\mathbf{W}$ is a weight matrix, defining a different weighted combination of input variables for each y_j. $\mathbf{b}$ is a constant term variably known as "bias", "threshold" or "intercept", depending on the context. The interpretation of the bias is not wholly straightforward, but consider the case where each input variable x_i has zero mean; in this case, b_j represents the mean of y_j.

This model is known as *linear regression*, or multivariate linear regression in the case of $m > 1$. The model defines an n-dimensional hyperplane for each y_j, independently of the other output variables. y_j increases linearly along a steepest direction given by $\frac{\partial y_j}{\partial \mathbf{x}} = (w_{1j}, w_{2j}, \ldots, w_{nj})$.

A graphical illustration of the model is provided in figure 2.4. We consider the model to be composed of two "layers": the input layer, $\mathbf{x}$, and the output layer, $\mathbf{y}$. Each layer is additionally composed

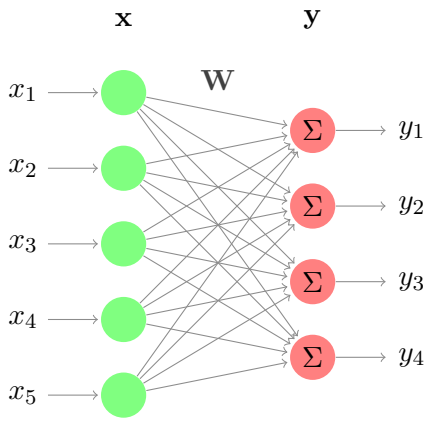

Figure 2.4: A simple linear model consisting of two layers. Each unit in the input layer corresponds to an input variable. Each output unit calculates a weighted sum of the input units.

of "units", corresponding to the individual scalar variables $x_1, \ldots, x_n, y_1, \ldots, y_m$. Every unit in the input layer is connected to every unit in the output layer, each connection representing a single weight. The output units perform a summation of the weighted input variables before adding a bias value, yielding the final output.

Linear regression can be generalised by inserting a so-called "activation function" g before outputting the final value:

$$\mathbf{y} = g(\mathbf{x}\mathbf{W} + \mathbf{b}). \tag{2.24}$$

If g is chosen to be the logistic function $g(\mathbf{z})_j = \sigma(z_j)$, this is known as *logistic regression*. The logistic function constrains the output to the range of $(0, 1)$, allowing y_j to be interpreted as modelling the Bernoulli distribution $P(j = 1 \mid \mathbf{x})$.

Stacked linear models

While the generalized model is powerful in its own right, it has fatal drawbacks for certain types of data. Consider the case of a one-dimensional output y given by $y = g(\mathbf{x} \cdot \mathbf{w} + b)$, where $\mathbf{w}$ is a weight vector. Let $\mathbf{v}$ be a vector orthogonal to $\mathbf{w}$, so that $\mathbf{v} \cdot \mathbf{w} = 0$. We can now see that $y = g((\mathbf{x} + \mathbf{v}) \cdot \mathbf{w} + b) = g(\mathbf{x} \cdot \mathbf{w} + \mathbf{v} \cdot \mathbf{w} + b) = g(\mathbf{x} \cdot \mathbf{w} + b)$. In other words, **linear models can only discriminate between data points along the axis defined by the weight vector**. As a result of this, the model's decision boundaries will all be linear and parallel, meaning that it will only be able to classify data that is *linearly seperable*. The problem is illustrated in figures 2.5(a)–(c).

However, we can extend our model to enable it to handle more complex data, by stacking several linear models on top of each other. This is done by inserting a new layer, called a "hidden" layer, between the input and output layers. Let $\mathbf{y}^0 = \mathbf{x}$ be the input layer, $\mathbf{y}^1$ the hidden layer, and $y^2 = y$ the output layer. We let

$$y_1^1 = 2\varphi(y_1^0) - 1 \tag{2.25}$$
$$y_2^1 = 2\varphi(y_2^0) - 1 \tag{2.26}$$
$$y^2 = y_1^1 + y_2^1 - 0.5 \tag{2.27}$$

where $\varphi(x) = \exp(-x^2)$. As can be seen in figures 2.5(d)–(f), this allows us to solve our classification problem.

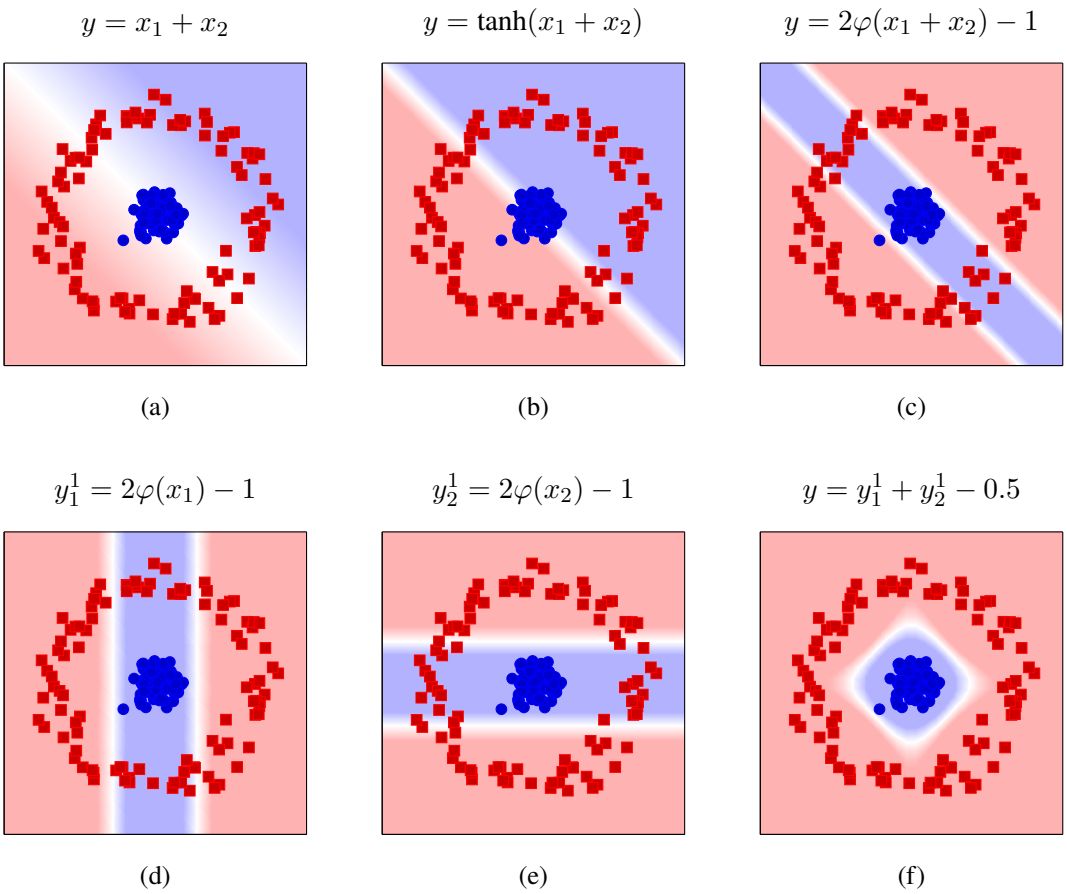

Figure 2.5: A classic classification problem. Using our model we wish to correctly classify the blue and red data points. We take the output of our model to mean blue if it is positive, and red otherwise. (a) The model defines a plane in three-dimensional space. The model can only influence the direction, slant and position of the plane. (b) A squashing function limits the range of the output to -1 to 1. (c) Using a non-monotonic activation function (here the Gaussian function $\varphi(x) = \exp(-x^2)$) it is possible to achieve more than one decision boundary, though all decision boundaries will be linear and parallel. However, by inserting an extra "hidden" layer between the input and output layers, the decision boundary can be made more complex. (d)–(e) The output of the units in the hidden layer. (f) Using a linear combination of the hidden units, the resulting decision boundary perfectly separates the two classes.

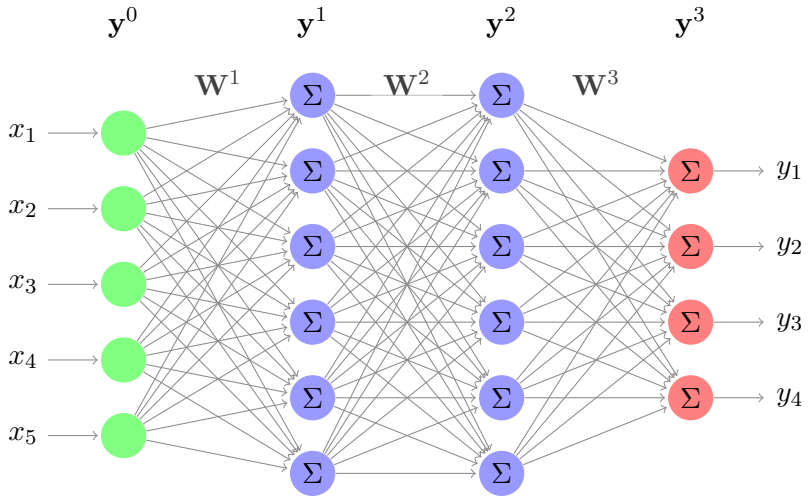

Figure 2.6: A feedforward neural network with two hidden layers. As the data is moved through the hidden layers it is gradually transformed into a representation that hopefully enables the problem to be solved by the final linear model.

In general, our stacked model can be defined as

$$\mathbf{y}^0 = \mathbf{x} \tag{2.28}$$
$$\mathbf{y}^l = g^l(\mathbf{y}^{l-1}\mathbf{W}^l + \mathbf{b}^l) \quad l \in [1, N] \tag{2.29}$$
$$\mathbf{y} = \mathbf{y}^N \tag{2.30}$$

where N is the number of hidden or output layers, and g^l, $\mathbf{W}^l \in \mathbb{R}^{n^{l-1} \times n^l}$, $\mathbf{b}^l \in \mathbb{R}^{1 \times n^l}$ and n^l are the activation function, weight matrix, bias vector and layer size (in number of units) corresponding to layer l, respectively. This is known as a *feedforward artificial neural network*; see figure 2.6 for a graphical representation. By setting

$$g^1(\mathbf{z})_i = 2\varphi(z_i) - 1 \qquad \mathbf{W}^1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad \mathbf{b}^1 = \mathbf{0} \tag{2.31}$$
$$g^2(z) = z \qquad \mathbf{W}^2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \qquad b^2 = -0.5 \tag{2.32}$$

with $N = 2$, we obtain the previous example.

The number of hidden layers to use in a network deserves some consideration. It has been shown that a feedforward network with a single hidden layer can approximate virtually any continuous function, as long as the number of hidden units is large enough (Hornik et al. 1989). However, theoretical results suggest that the complexity of the model in terms of the types of functions it is able to express grows exponentially in the number of layers, meaning that a shallow network with only one hidden layer would need to consist of an exponential number of hidden units to match the complexity of a deep architecture (Montúfar et al. 2014).

Activation functions

It is possible to use different activation functions for different layers, or even different units within the same layer. Typically, however, all hidden layers use the same activation function, with the activation function chosen for the output layer depending on the task at hand (e.g. classification, regression).

Output layers

For classification, an activation function that makes it possible to interpret the output as a one or more probability distributions is generally used. A common choice is the softmax function, which normalises the output to sum to 1:

$$\text{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}}. \tag{2.33}$$

For multi-label classification one can use the logistic function, defined as

$$\sigma(\mathbf{z})_i = \frac{1}{1 + e^{-z_i}}. \tag{2.34}$$

When doing regression on the other hand, it is common to simply use a linear activation function:

$$g(\mathbf{z}) = \mathbf{z}. \tag{2.35}$$

Hidden layers

It can be shown that a stacked model with linear activations in the hidden layers is equivalent to a shallow model with no hidden layers. Thus, it is vital that the activation function used for the hidden layers be non-linear. However, it can be advantageous to use a *mostly linear* activation function, not least because both the function itself and its derivative becomes very fast to calculate.

Examples of mostly linear activation functions that have been shown to outperform other activation functions in many contexts include the rectified linear unit (Glorot et al. 2011):

$$\text{ReLU}(\mathbf{z}) = \max(\mathbf{0}, \mathbf{z}) \tag{2.36}$$

and the maxout unit (Goodfellow et al. 2013):

$$\text{maxout}(\mathbf{y}^{l-1}) = \max(\mathbf{y}^{l-1}\mathbf{W}^l + \mathbf{b}^l, \mathbf{y}^{l-1}\mathbf{V}^l + \mathbf{c}^l) \tag{2.37}$$

where max is performed element-wise, i.e. $\max(\mathbf{u}, \mathbf{v})_i = \max(u_i, v_i)$. Note that the input to maxout is the output of the previous layer rather than a weighted sum; it is a generalisation of the ReLU, keeping multiple (not necessarily limited to only two) sets of weight and bias values.

add source (german??) or remove as digression

weights play a large role too...

Historically it has also been common to use sigmoidal (S-shaped) functions such as the logistic function or tanh for hidden layers as well, as a way to approximate the spiking behaviour of biological neurons. However, a major issue with a sigmoidal activation function is that, in addition to being comparatively slow to compute, the derivative of the function quickly falls towards 0 as the function saturates for large positive and negative input values, which tends to occurr as training progresses. This causes training of the network to slow down, especially in networks with many hidden layers where the derivatives of multiple activation functions are multiplied, in a phenomenon known as the “vanishing gradient” problem.

Training neural networks

In order to produce any useful results, the parameters (weights and biases) $\boldsymbol{\theta}$ of the network need to be tuned. This tuning is referred to as *training* the network, and is performed by minimising some loss function (also known as a cost function or objective function) $L(\boldsymbol{\theta}; \mathbf{x}, \mathbf{y})$ where $\mathbf{x}$ is the input to the

network, and $\mathbf{y}$ is the *expected* output of the network. If $\hat{\mathbf{y}} = f(\mathbf{x}; \boldsymbol{\theta})$ is the actual output of the network, $L(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta})$ defines some measure of how dissimilar $\hat{\mathbf{y}}$ and $\mathbf{y}$ are.

For classification, the most common loss function is the cross-entropy between $\mathbf{y}$ and $\mathbf{x}$, defined as

$$L_{\text{CE}}(\boldsymbol{\theta}; \mathbf{x}, \mathbf{y}) = -\sum_{j=1}^{m} y_j \log \hat{y}_j. \tag{2.38}$$

For single-class classification such that one output is 1 and all others 0, this simplifies to

$$L_{\text{CE}}(\boldsymbol{\theta}; \mathbf{x}, \mathbf{y}) = -\log \hat{y}_j \tag{2.39}$$

for the j such that $y_j = 1$. Minimising the cross-entropy between $\mathbf{y}$ and $\mathbf{x}$ with respect to $\mathbf{x}$ is equivalent to minimising the Kullback–Leibler divergence between the same, making it appropriate as a loss function when interpreting the network output as a probability distribution.

Regression often makes use of the mean squared error (sometimes multiplied by a factor of $\frac{1}{2}$) instead:

$$L_{\text{MSE}}(\boldsymbol{\theta}; \mathbf{x}, \mathbf{y}) = \|\hat{\mathbf{y}} - \mathbf{y}\|_2^2. \tag{2.40}$$

Autoencoders, which attempt to find a low-dimensional representation of the input data from which the data can be reconstructed, use $L_{\text{MSE}}(\boldsymbol{\theta}; \mathbf{x}, \mathbf{x})$, called the reconstruction error, as the loss function.

Once the loss function is defined, the network can be trained by iteratively modifying the network parameters through gradient descent. Usually a special form of gradient descent called minibatch gradient descent is used, where the network is presented with a small subset of the examples at each iteration, and the gradient is averaged over the presented examples. Let $B = \{i_1, i_2, \ldots, i_K\}$ be K indices forming one minibatch. We can then state the training procedure as

$$\boldsymbol{\theta}^{t+1} \leftarrow \boldsymbol{\theta}^t - \eta \frac{\partial}{\partial \boldsymbol{\theta}} \left(\frac{1}{|B|} \sum_{i \in B} L(\boldsymbol{\theta}^t; \mathbf{x}_i, \mathbf{y}_i) \right) \tag{2.41}$$

where η is the "learning rate". Tuning the learning rate is important, as a high learning rate means faster learning, but setting it too high may prevent the training from converging.

By exploiting the layered structure of feedforward neural networks, it is possible to calculate the gradient efficiently in a recursive manner. This algorithm is commonly referred to as the backpropagation algorithm (Rumelhart et al. 1986).

Chapter 3

Related work

This chapter provides a brief overview of recent research into unsupervised acoustic modelling. The approaches discussed here can broadly be divided into two categories: bottom-up approaches that infer the acoustic model directly from the speech frames, and top-down approaches that first segment the speech into syllable- or word-like units, and afterwards try break these units into smaller subword units.

Bottom-up approaches

As an individual speech frame only make up a fraction of a complete speech sound, it is natural to model and segment the speech using a model that can capture time dependencies, such as a hidden Markov model (HMM), rather than attempt to cluster the speech frames directly. One issue with this approach, however, is that the number of possible states (i.e. subword units) is unknown a priori.

Varadarajan et al. (2008) tackle this problem by first defining a one-state HMM, and then iteratively splitting and merging states as needed to account for the data according to a heuristic. Training stops once the size of the HMM reaches a threshold. After training, each state in the HMM can be thought to correspond to some allophone (context-dependent variant realisation) of a phoneme. It should be noted, however, that in order to interpret a given state sequence as a single phoneme, Varadarajan et al. train a separate model using labelled speech to perform this mapping. The method is thus not fully unsupervised.

Lee and Glass (2012) take a fully probabilistic approach, defining a model that jointly performs segmentation and acoustic modelling. An infinite mixture model of tri-state HMM-GMMs modelling subword units is defined using the Dirichlet process, and latent variables representing segment boundaries are introduced. The data can be thought to be generated by repeatedly sampling an HMM to model a segment, sampling a path through the HMM, and for each state in the path sampling a feature vector from the corresponding GMM. The probability of transitioning from one unit to another is thus not modelled. Inference of the model is done using Gibbs sampling.

Siu et al. (2014) use an HMM of a more classic form to model the data. An initial transcription of the data in terms of state labels is first generated in an unsupervised manner using a segmental GMM (SGMM). The HMM and transcription are then iteratively updated, maximising the probability of the model parameters given the transcription, and the transcription given the model parameters. Note that the number of allowed states are here defined in advance. n-gram statistics are then collected from the transcription and used for tasks such as unsupervised keyword discovery.

Diverging from previous approaches using temporal models, Chen et al. (2015) perform standard clustering of speech frames using an infinite Gaussian mixture model. After training, the speech frames are represented as posteriorgrams, which have been shown to be more speaker-invariant than

other features such as MFCCs (Zhang and Glass 2010). Despite the simple approach, this turned out to be the overall best-performing model in the first track of the 2015 Zero Resource Speech Challenge (Versteegh et al. 2016). Heck et al. (2016) later further improved on the model by performing clustering in two stages, with an intermediate supervised dimensionality reduction step using the clusters derived from the first clustering step as target classes.

Synnaeve and Dupoux (2016) use a siamese network to create an embedding where speech frames close to each other are considered to belong to the same subword unit, while distant speech frames are said to differ. A siamese network is a feedforward neural network that takes two inputs and adjusts its parameters to either maximise or minimise the similarity of the corresponding outputs (Bromley et al. 1994).

Top-down approaches

remove information already included in the introduction

Top-down approaches start by first finding pairs of longer word-like segments using unsupervised term discovery (UTD). This information provides constraints that can be use to find speech frame representations that are more stable within a given subword unit. The rationale is that while at the frame level the same speech sound can seem quite different between different speakers or even different realisations of the sound by the same speaker, patterns over a longer duration of time are easier to identify; this idea is illustrated in Jansen et al. (2013).

The UTD systems used in this context are generally based on the segmental dynamic time warping (S-DTW) developed by Park and Glass (2008). S-DTW works by repeatedly performing DTW on two audio streams while constraining the maximum amount of warping allowed, each time changing the starting point of the DTW in both streams. This yields a set of alignments, from which the stretches of lowest average dissimilarity in each alignment can be extracted. Unfortunately, this approach is inherently $O(n^2)$ in time. To remedy this, Jansen and Van Durme (2011) introduced an approximate version that uses binary approximations of the feature vectors to perform the calculations in $O(n \log n)$ time using sparse similarity matrices; this system also serves as the baseline for the second track of the Zero Resource Speech Challenge (Versteegh et al. 2015).

Jansen and Church (2011) describe a method for finding subword units, assuming that clusters corresponding to words, each cluster containing multiple examples of that word in the form of audio, are given. For each word, an HMM is trained on all the corresponding examples, the number of states in the model being set to a number proportional to the average duration of the word. The states from each HMM are then collected and clustered based on the similarity of their distributions, forming clusters that hopefully correspond to subword units.

Jansen et al. (2013) take somewhat of an inverse approach, starting by clustering the whole data on a frame level, with the assumption that each cluster will tend to correspond to some speaker- or context-dependent subword unit. They then look at pairs of word-like segments known to be of the same type and calculate how often clusters tend to co-occurr. The clusters are then partitioned so that clusters that co-occurr often are placed in the same partition.

Synnaeve et al. (2014) introduce a neural network known referred to as the ABnet, based on siamese networks (Bromley et al. 1994). The network takes a pair of speech frames as input, and adjusts its parameters so that the outputs are collinear if the inputs are known to correspond to the same subword unit, and orthogonal otherwise, using a cosine-based loss function. Thiolliere et al. (2015) made use of this approach in the Zero Resource Speech Challenge, also incorporating unsupervised term discovery so as to make the whole process unsupervised, yielding competitive results (Versteegh et al. 2016).

Zeghidour et al. (2016) experiment with supplying the ABnet with scattering spectrum features instead of filter bank features, showing that with the right features, a shallow architecture may outperform a deep architecture, especially when the amount of available data is low.

Kamper et al. (2015) use an autoencoder-like structure, where a neural network is trained to "reconstruct" a frame given another frame known to be of the same type. Renshaw et al. (2015) used this architecture in the Zero Resource Speech Challenge, albeit with a deeper decoder.

This thesis 

make description more application agnostic?

Two of the most successful approaches so far are the clustering approach of Chen et al. (2015) and the siamese network approach of Thiolliere et al. (2015). We pose the question of whether it is possible to combine the two approaches by first clustering the data in an unsupervised manner using a probabilistic model, and then improving the resulting posteriorgrams using speech fragment information. This way we are able to take advantage of both the whole unlabelled data set, and the smaller set of discovered fragments.

Many probabilistic models, such as Gaussian mixture models and hidden Markov models, have a concept of latent states or classes. We pose the problem of improving posteriorgrams from such a model as one of merging, or partitioning, these classes. By first training the model in a fully unsupervised manner, it learns classes that can generally be assumed to be highly speaker-specific. We can then use weak supervision to merge these classes, yielding representations that are more speaker invariant.

A partitioning of classes can be viewed as a surjection from the original set of classes to a class set of lower cardinality, but finding this surjection is a discrete problem which is difficult to optimise for. However, a benefit of posteriorgrams is that the probability of an output class can be described as a simple sum of the probabilities of the classes that map to the class in question. This means that the surjection can be approximated using a continuous linear model which can be optimised through standard gradient descent. A linear model also has the added benefit of being more interpretable than deep networks such as that of Thiolliere et al. (2015). While the approach of partitioning posteriorgrams is very reminiscent of Jansen et al. (2013), the major difference is that in place of direct clustering of classes, we are instead trying to maximise the similarity/dissimilarity between pairs of speech fragments, which only indirectly results in a partitioning of the classes.

Chapter 4

Method

This chapter describes the shallow siamese network used to find an approximate class-merging surjection.

Model

We take as input a set $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$ of N pairs of M-dimensional posteriorgrams, i.e. (row) vectors of probabilities such that the probabilities sum to one. Additionally, we have a set of indicators $\{c_i\}_{i=1}^N$ such that c_i is 1 if $\mathbf{x}_i$ and $\mathbf{y}_i$ belong to the same class, and 0 otherwise. The posteriorgrams are taken to represent a distribution over M discrete "pseudo"-classes (e.g. allophones), where several pseudo-classes together describe a single "true" class (e.g. phonemes). Our goal is then to find a surjection that maps the M pseudo-classes to a smaller set of D classes, where we take the probability of a single output class to be the sum of the probabilities of the pseudo-classes that map to the class in question.

To simplify optimisation we relax the problem to one of instead finding a continuous linear mapping $f : [0,1]^M \rightarrow [0,1]^D$ from the original space to a lower-dimensional space, such that $f(\mathbf{x}_i)$ and $f(\mathbf{y}_i)$ are close if $\mathbf{x}_i$ and $\mathbf{y}_i$ belong to the same true class, and distant otherwise. We consider each output probability to be a weighted combination of input probabilities: $f(\mathbf{x})_j = \sum_{i=1}^M x_i w_{ij}$, or in matrix notation:

$$f(\mathbf{x}) = \mathbf{x}\mathbf{W} \tag{4.1}$$

where $\mathbf{W} = (w_{ij}) \in \mathbb{R}^{M \times D}$. If the elements of $\mathbf{W}$ are constrained to only take on values in $\{0, 1\}$, and each row of $\mathbf{W}$ contains exactly one element with the value 1, the problem is reduced to finding an exact surjection.

Even in the relaxed version of the problem, we need to put certain constraints on $\mathbf{W}$ in order to ensure that the output $f(\mathbf{x})$ is a posteriorgram. First, we need to ensure that all outputs are positive. As the input $\mathbf{x}$ is a posteriorgram, meaning that all elements in $\mathbf{x}$ are positive, it clearly suffices to ensure that all elements in $\mathbf{W}$ are positive. Second, the output probabilities must sum to 1. This can be achieved by ensuring that the elements of each row of $\mathbf{W}$ sum to 1, as can be seen by:

$$f(\mathbf{x})\mathbf{1}_D = \mathbf{x}\mathbf{W}\mathbf{1}_D = \mathbf{x}\mathbf{1}_D = 1 \tag{4.2}$$

where $\mathbf{1}_D$ is a column vector of D ones.

In order to ensure that these constraints hold, we construct our model as follows:

$$\mathbf{V} \in \mathbb{R}^{M \times D} \tag{4.3}$$

$$\widetilde{\mathbf{W}} = |\mathbf{V}| \tag{4.4}$$

$$\mathbf{W} = \widetilde{\mathbf{W}} \oslash \left(\widetilde{\mathbf{W}} \mathbf{1}_D \mathbf{1}_D^T\right) \tag{4.5}$$

$$f(\mathbf{x}) = \mathbf{x}\mathbf{W} \tag{4.6}$$

where $|\cdot|$ denotes the element-wise absolute value, and $\oslash$ denotes element-wise division. This formulation makes it possible to optimise the model while ensuring that the constraints on $\mathbf{W}$ hold, by performing gradient descent with respect to $\mathbf{V}$. Note that the absolute value is almost everywhere differentiable, and the non-differentiability at 0 does not matter in practice.

To encourage the model to place points belonging to the same class close together in the output space, we consider the model as a siamese network. Conceptually this involves duplicating the model, creating two identical copies of the same network, with the parameters shared. We then feed one input each to both copies, and calculate the loss function using the corresponding outputs:

$$L(\mathbf{V}; \mathbf{x}, \mathbf{y}, c) = \begin{cases} D_{\text{same}}(f(\mathbf{x}; \mathbf{V}), f(\mathbf{y}; \mathbf{V})) & \text{if } c = 1 \\ D_{\text{diff}}(f(\mathbf{x}; \mathbf{V}), f(\mathbf{y}; \mathbf{V})) & \text{if } c = 0 \end{cases} \tag{4.7}$$

where D_{same} and D_{diff} are the dissimilarity/similarity measures for pairs belonging to the same class, and pairs belonging to different classes, respectively. The loss function over a minibatch B is given by the average

$$\frac{1}{|B|} \sum_{i \in B} L(\mathbf{V}; \mathbf{x}_i, \mathbf{y}_i, c_i) \tag{4.8}$$

which is minimised with respect to $\mathbf{V}$.

Loss function

As the output of the model is a probability distribution, it makes intuitive sense to use a statistical divergence as a measure of similarity. Perhaps the most well-known divergence is the Kullback-Leibler (KL) divergence, defined as:

$$\text{KL}(\mathbf{x}||\mathbf{y}) = \sum_i x_i \log_2 \frac{x_i}{y_i}, \tag{4.9}$$

where we take $0 \log_2 0$ to be 0. The KL divergence is always positive, and is 0 only if $\mathbf{x} = \mathbf{y}$. However, it is unbounded, and undefined if there is an i such that $y_i = 0$ but $x_i \neq 0$. As such, trying to maximise the dissimilarity between two distributions with respect to the KL divergence is an ill-posed problem, as this will force the divergence to tend towards infinity.

A better choice is the Jensen-Shannon (JS) divergence, defined as

$$\text{JS}(\mathbf{x}||\mathbf{y}) = \frac{1}{2}\text{KL}(\mathbf{x}||\mathbf{m}) + \frac{1}{2}\text{KL}(\mathbf{y}||\mathbf{m}) \tag{4.10}$$

where $\mathbf{m} = (\mathbf{x} + \mathbf{y})/2$. The JS divergence is always defined, and is bounded between 0 (for identical distributions) and 1 (for distributions with disjoint support), assuming that the base 2 logarithm is used. Additionally, the square root of the JS divergence is a metric satisfying the triangle inequality (Endres and Schindelin 2003); here we make use of this fact, in the hope that the metric properties will result in a more well-behaved loss function.

Thus, we define the loss function as

$$L_{\text{JS}}(\mathbf{V};\mathbf{x},\mathbf{y},c) = \begin{cases} \sqrt{\text{JS}(f(\mathbf{x};\mathbf{V})||f(\mathbf{y};\mathbf{V}))} & \text{if } c = 1 \\ 1 - \sqrt{\text{JS}(f(\mathbf{x};\mathbf{V})||f(\mathbf{y};\mathbf{V}))} & \text{if } c = 0, \end{cases} \tag{4.11}$$

thereby minimising the root JS divergence between pairs belonging to the same class, and maximising the divergence between pairs belonging to different classes[1].

Entropy penalty

To make the output of the model interpretable, it is desirable to ensure that for a given input, only one output unit is active. This can be done by introducing an entropy penalty, which attempts to minimise the spread of the probability mass. The entropy of a probability vector $\mathbf{x} = (x_1, \ldots, x_D)$ is defined as

$$H(\mathbf{x}) = -\sum_{i=1}^{D} x_i \log_2 x_i. \tag{4.12}$$

However, this definition is sensitive to the value of D; for instance, the entropy of a uniform distribution vector is $\log_2 D$.

As we may wish to vary the number of outputs of the model, it is of interest for the entropy penalty to be invariant to the number of outputs. We therefore introduce the normalised entropy, defined as

$$\hat{H}(\mathbf{x}) = \frac{1}{\log_2 D} H(\mathbf{x}). \tag{4.13}$$

The normalised entropy is always between 0 (for degenerate distributions) and 1 (for uniform distributions).

The entropy penalty implicitly encourages sparsity in $\mathbf{W}$, as the only way to avoid spreading the probability mass across several outputs is for each row of $\mathbf{W}$ to only contain a single element close to 1. It is thus through this penalty that we enforce the model to find an approximate surjection. In summary, our final loss function over a minibatch B is as follows:

$$L(\mathbf{V};B) = \frac{1}{|B|}\sum_{i\in B} L_{\text{JS}}(\mathbf{V};\mathbf{x}_i,\mathbf{y}_i,c_i) + \frac{\lambda}{2|B|}\sum_{i\in B}(\hat{H}(f(\mathbf{x}_i;\mathbf{V})) + \hat{H}(f(\mathbf{y}_i;\mathbf{V}))) \tag{4.14}$$

where λ is a hyperparameter.

[1] For identical or near-identical $\mathbf{x}$ and $\mathbf{y}$, the JS divergence may become negative due to rounding errors caused by limited floating point precision; this can be counteracted by adding a small constant value before taking the square root.

Chapter 5

Experiments

This chapter describes the application of the model described in chapter 4 to the task of unsupervised modelling of speech, and in particular the use of the model to improve posteriorgrams generated from a Gaussian mixture model. First the experimental setup is described, including the data used, the how the data is processed, and how the models are implemented. Next, a number of experiments aimed at tuning hyperparameters and comparing models are described. Finally, the models are evaluated using the minimal-pair ABX task.

Experimental setup

Data

The 2015 Zero Resource Speech Challenge makes use of two corpora: The Buckeye corpus of conversational English (Pitt et al. 2007) and the NCHLT speech corpus of read Xitsonga (Barnard et al. 2014). For the challenge only a subset of the data is used, consisting of 12 speakers for a total of 5 hours of data for the Buckeye corpus, and 24 speakers for a total of 2.5 hours of data for the NCHLT Xitsonga corpus. Additionally provided is voice activity information indicating segments containing clean speech, as well as labels indicating the identity of the speaker.

Generating the posteriorgrams

MFCCs features were extracted from the data using a frame window length 25 ms which was shifted 10 ms for each frame, an FFT resolution of 512 frequency steps, and 40 mel-spaced triangular filter banks. 13 coefficients with both delta and delta-delta features were used. The MFCCs corresponding to segments with voice activity were clustered using an implementation of a Gaussian mixture model (GMM) provided by scikit-learn (Pedregosa et al. 2011). The GMM was trained using the expectation maximisation algorithm, using $M = 1024$ Gaussians with diagonal covariance matrices, for a maximum of 200 iterations. After training the posteriorgram for the nth frame is constructed as $\mathbf{p}_n = (p_n^1, p_n^2, \ldots, p_n^{1024})$ where $p_n^i = p(z_i \mid \mathbf{x}_n)$ is the posterior probability of the ith class given the nth frame.

z_i^n or z_i better?

Unsupervised term discovery

Pairs of similar speech fragments were discovered using the system developed by Jansen and Van Durme (2011), which serves as a baseline for the second track of the Zero Resource Speech Challenge. The system works by calculating the approximate cosine similarity between pairs of frames of

two input audio segments, based on discretised random projections of PLP features. For efficiency only frames found using an approximate nearest neighbour search are compared, yielding a sparse similarity matrix. Stretches of similar frames are then found by searching for diagonals in the similarity matrix, which which are then aligned using dynamic time warping (DTW). Pairs of segments with a DTW score above a certain threshold are kept and clustered based on pairwise DTW similarity, resulting in a set of clusters of speech segments, or fragments, thought to be of the same class (e.g. word).

This process yielded 6512 fragments and 3149 clusters for the Buckeye corpus, and 3582 fragments and 1782 clusters for the NCHLT Xitsonga corpus[1]. For each cluster every possible pair of fragments was extracted from the collection of posteriorgrams retrieved from the GMM and aligned using DTW, yielding pairs of speech frames belonging to the same class. Let K be the total number of pairs of fragments aligned. To generate a set of pairs of frames belonging to different classes, K fragments were sampled uniformly from the full collection of fragments. For each such fragments, another fragment was sampled uniformly from the fragments belonging to a different cluster. When sampling fragments belonging to a different cluster, the sampling was performed using only either fragments spoken by the same speaker, or fragments spoken by a different speaker, with a probability corresponding to the ratio of same-speaker to different-speaker pairs among the same-class fragment pairs. The different-class fragment pairs were aligned by simply truncating the longer fragment.

70% of the same-class and different-class fragment pairs were used for training, with the remaining pairs used for validation to determine when to interrupt the training of the models.

Model implementation

mention server specification? cpu/ram

We used $D = 64$ outputs for all models. The models were trained using AdaMax (Kingma and Ba 2014) with the recommended default parameters $\alpha = 0.002$, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. All frames used for training were shuffled once at the start of training, and a minibatch size of 1000 frames was used. The models were trained until no improvement had been observed on a held-out validation set for 15 epochs, where one epoch is defined as one complete scan over the training data.

All network models were implemented in Python 3.5 using Theano (Al-Rfou et al. 2016) for automatic differentiation and GPU acceleration, librosa (McFee et al. 2017) for feature extraction, scikit-learn (Pedregosa et al. 2011) for various utilities, and numba (Lam et al. 2015) for accelerating various code, in particular dynamic time warping.

Tuning the entropy penalty

The entropy penalty λ is a free parameter, which is data dependent and must be manually specified. Ideally, λ should be such that the entropy is reduced to a satisfactory degree, without sacrificing the Jensen-Shannon loss. As both the normalised entropy loss and the Jensen-Shannon loss are bounded between 0 and 1, one might expect the optimal value of λ to be in the vicinity of 1. We train models using $\lambda \in \{0, 0.05, 0.1, \ldots, 0.95\}$ for both the Buckeye and NCHLT Xitsonga corpora.

The final validation errors for each model are reported in figure 5.1. For both corpora, the entropy drops quickly even for small λ, suggesting that the entropy is relatively easy to optimise for. As the entropy penalty is increased, the entropy itself does not decrease; however, the different-class JS loss decreases at the expense of the same-class JS loss. For future experiments, a penalty of $\lambda = 0.1$ is used.

why?

[1] The cluster files used for this work were generously provided by Roland Thiollière and Aren Jansen.

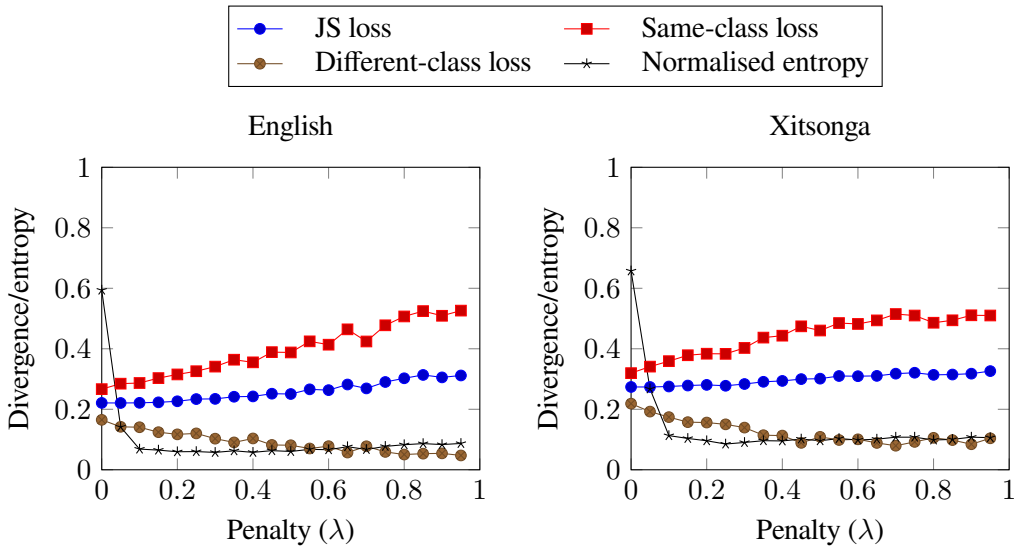

Figure 5.1: Effect of varying the entropy penalty for the English (left) and Xitsonga (right) corpora. The average entropy of the output distribution over the validation samples is shown along with the (root) Jensen-Shannon loss: Both the combined JS loss that is optimised for, and separately for same-class and different-class frame pairs.

Balancing same-class and different-class losses

When enforcing low entropy in the output distribution, the resulting weight matrix becomes sparse. For instance, after training the model with $\lambda = 0.1$, and inspecting the row-normalised matrix $\mathbf{W}$, we find that the largest element on each row is close to 1: on average across the 1024 rows 0.98 for English and 0.92 for Xitsonga. We can thus inspect $\mathbf{W}$ to see how many of the 64 outputs are actually being used by the model. We take the sum over each column of $\mathbf{W}$. This sum describes roughly how many inputs are mapped to each output. We find that for both English and Xitsonga, this sum is above 0.5 for only a minority of outputs: 11 outputs for English, and 10 outputs for Xitsonga. For English, where $\mathbf{W}$ is particularly sparse, none of the other 53 sums even reach 0.05.

Thus, it seems that the entropy penalty naturally encourages the model to make use of only a subset of the outputs. However, the actual number of outputs used is not realistic in terms of how many phonemes one would expect to find in a language; it seems that the same-class loss is forcing too many input classes to merge. To solve this, we restate the Jensen-Shannon loss function, allowing us to specify how much relative weight to give to the same-class and different-class losses. Let $B_1 = \{i \in B : c_i = 1\}$ be the subset of same-class frame pairs in the current minibatch, and $B_0 = \{i \in B : c_i = 0\}$ the subset of different-class frame pairs. We then restate the loss as

$$\frac{1}{(\alpha+1)|B_1|}\sum_{i\in B_1} L_{\text{same}}(\mathbf{V};\mathbf{x}_i,\mathbf{y}_i) + \frac{\alpha}{(\alpha+1)|B_0|}\sum_{i\in B_0} L_{\text{diff}}(\mathbf{V};\mathbf{x}_i,\mathbf{y}_i), \tag{5.1}$$

where L_{same} and L_{diff} are the same-class and different-class losses defined in equation (4.11). α is a hyperparameter specifying how much more to weight the different-class loss over the same-class loss.

α needs to be carefully tuned: A too small α will cause too many input classes to merge, including classes that correspond to completely different phonemes, while a too large α will cause input classes that do correspond to the same phoneme to fail to merge. In order to find a good value for α, without making use of the gold transcription or prior knowledge of the number of phonemes present in the lan-

guages in question, we make use of the fragment clusters discovered by the unsupervised term discovery system. The intuition is that the goal of our model is to push apart different clusters, while keeping fragments within a cluster as similar as possible. To measure the success of our model, then, we can make use of a cluster separation measure.

Here we use the silhouette (Rousseeuw 1987), which makes use of the average similarity between a sample and every other sample in the same cluster, and between a sample and every sample in the most similar other cluster. The silhouette ranges from -1 to 1, with a value close to 1 indicating that the clusters are well separated. Models were trained for $\alpha \in \{1, 1.5, 2, 2.5, 3, 3.5, 4\}$, with an entropy penalty of $\lambda = 0.1$. The silhouette was then calculated on a subset of 1000 of the fragment clusters, using the output of the trained models to represent the frames of the fragments. The similarity between to fragments was calculated as the DTW score using the symmetrise Kullback-Leibler divergence as a similarity measure between individual frames.

is the below paragraph uninteresting? a bit too model-specific?

To easily get an estimate of the number of outputs used by the model, we also define the "spread" of the model as follows. We take the average of the jth column:

$$q_j = \frac{1}{M} \sum_{i=1}^{M} w_{ij}. \tag{5.2}$$

This represents the average mapping to the ith output. As $\mathbf{W}$ is row-normalised, the elements of $Q = (q_1, q_2, \ldots, q_D)$ sum to 1, and we can thus treat Q as describing a probability distribution. A uniform distribution means that each output has the same number of inputs mapped to it. Now consider the case where there are K outputs such that the same number of inputs maps to each output, while no inputs map to any other outputs. The normalised entropy of Q is then given by

$$\hat{H}(Q) = -\frac{1}{\log_2 D} \sum_{i=1}^{K} \frac{1}{K} \log_2 \frac{1}{K} = \frac{\log_2 K}{\log_2 D}. \tag{5.3}$$

Solving for K we have

$$K = D^{\hat{H}(Q)}, \tag{5.4}$$

which is an approximation of the number of outputs used by the model, which we define as the spread. A value of K close to D is an indicator that all the outputs are being used equally, suggesting that it may be a good idea to increase the number of outputs.

Figure 5.2 shows the silhouette and spread for different values of α. As one might expect, more emphasis on the different-class loss results in a higher spread, i.e. a larger number of output classes. The optimal value of α seems to be around 1.5 for both data sets, we use this value of α going forward.

Discretising the model

As the resulting model is sparse, we can retrieve an exact surjection by discretising the model. We do this by for each row in $\mathbf{W}$ setting the largest element to 1 and the remaining elements to 0. Using the discretised model as a base, we additionally experiment with discretising the output distribution by setting the largest output to 1 and the rest to 0; this can be thought of as taking the argmax of the output distribution.

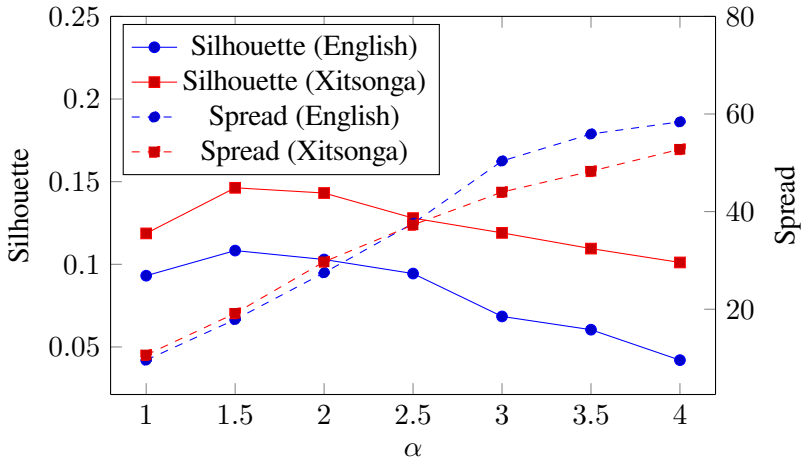

Figure 5.2: Silhouette and spread for different weightings of the same-class and different-class losses.

Comparison with deep models

To get an idea of how the JS loss performs in general, we build a deep network with two hidden layers of 500 sigmoid units each, with 64 softmax outputs. The network is trained using the non-rebalanced JS loss. As softmax outputs are naturally sparse, we do not enforce any entropy penalty. For comparison we train the same architecture, albeit with sigmoid outputs instead, using the coscos^2 loss of Synnaeve et al. (2014). This is the architecture used by Thiolliere et al. (2015) in the 2015 Zero Resource Speech Challenge.

As input to both networks we use the log-scale outputs of 40 mel-scaled filter banks. All other relevant parameters are the same as for the MFCCs calculated in section 5.1.2. The filter bank outputs are normalised over the whole data set to have zero mean and unit variance for all dimensions. Each frame is fed to the network with a context of 3 frames on both sides, for a total of 280 values used as input to the network. All fragments are DTW aligned and sampled as in section 5.1.3.

Interpreting the model

ABX evaluation

We evaluate the models discussed on the minimal-pair ABX task (Schatz et al. 2013). In the task we are presented with three speech fragments A, B and X, where A and B form minimal pairs, i.e. they only differ by a single phoneme. The task is to decide which of either A or B belongs to the same category as X. This is done by DTW-aligning A and B with X with respect to some underlying frame-based metric. The fragment closest to X according to the DTW score is chosen. The task takes two forms: within-speaker discriminability, where all fragments belong to the same speaker, and across-speaker discriminability, where A and B belong to one speaker while X belongs to another.

The models are evaluated using a evaluation toolkit provided for the Zero Resource Speech Challenge. The results are shown in table 5.1, along with the silhouette for each model. The frame-based metric is chosen as the symmetrised Kullback-Leibler divergence (with the model output normalised as necessary), with the exception of the model with discretised output, which uses the cosine distance, which for one-hot vectors amounts to a distance of 0 for identical and 1 for non-identical vectors.

We can see that in general, the silhouette seems to be indicative of the relative performance on the ABX task. Our suspicion that the number of outputs used were too few when using the Jensen-

	English			Xitsonga		
Model	Silhouette	Within	Across	Silhouette	Within	Across
GMM posteriors	0.008	12.313	23.841	0.066	11.434	23.181
Non-rebalanced	0.089	14.195	21.369	0.111	16.477	25.551
Rebalanced	0.108	12.770	19.831	0.146	13.990	23.202
Discretised $\mathbf{W}$	0.124	12.013	19.261	0.170	12.702	21.888
Discretised output	0.010	16.513	24.565	0.014	19.404	29.150
Deep JS	-0.370	22.376	28.233	-0.320	18.190	24.759
Deep coscos^2	0.187	12.294	19.561	0.174	11.934	19.052

Table 5.1: Within-speaker and across-speaker ABX scores as well as the silhouette for the different models for both the English and Xitsonga data sets. GMM posteriors is the posteriorgrams extracted from the 1024-component Gaussian mixture model; non-rebalanced is the original loss presented in equation (4.14); rebalanced is the alternative loss presented in equation (5.1) with $\alpha = 1.5$; discretised $\mathbf{W}$ and discretised output are the models presented in section 5.4; and the deep models are those presented in section 5.5. The silhouette is calculated on a subset of 1000 clusters for each language. All shallow models are trained with an entropy penalty of $\lambda = 0.1$.

Shannon loss as originally stated is validated, with the rebalanced loss performing better for both English and Xitsonga. The performance of the model with discretised weights further suggests that the basic premise of improving posteriorgrams by partitioning is a sound one.

The deep model performs poorly when trained with the Jensen-Shannon loss, despite the same architecture performing well when trained with the coscos^2 loss. Inspecting the average output of the deep model over the English data set, we found that only 6 outputs are actually used by the model. This suggests that the JS loss is more sensitive than the coscos^2 loss when it comes to balancing the same-class and different-class losses.

Chapter 6

Discussion and conclusion

We end the thesis with some general remarks and ideas for future work.

Discussion

We have seen that the model is indeed able to improve on the input posteriors. In particular, the model improves the across-speaker performance, with little to no degradation of the within-speaker performance. However, the Jensen-Shannon loss function used is shown to perform worse in general than coscos^2, possibly as a result of being more sensitive to the balancing of the same-class and different-class losses. This can be explained by the fact that the Jensen-Shannon divergence is not directly interpretable—for instance, it is not clear that a same-class loss of 0.1 is as good as a different-class loss of 0.9. On the other hand, the cosine difference is more readily (geometrically) interpretable.

However, the model itself does come with a number of advantages over deep models. The linear nature of the model means that the number of parameters is small, making the model fast and easy to train, and robust against overfitting. This is especially the case when imposing the entropy penalty, which can be seen as restricting the capacity of the model. The sparsity of the model additionally makes it more interpretable, providing insight into how exactly the input classes are mapped to the output. The model is also readily convertible into an exact surjection, resulting in a proper partition of the input classes.

Another feature of the model is that it can take any kind of probability distribution as input, with the only requirement being that the underlying true classes are disentangled in the input. This makes it possible to use any kind of probabilistic model that admits a discrete posterior distribution over classes or states, including e.g. Gaussian mixture models or hidden Markov models. The resulting posteriorgrams can then be improved further by using the model to find a mapping to a smaller number of classes.

One important question is how sensitive the model is to the dimensionality of the input. As the model requires evidence in terms of same-class or different-class pairs to know where to map each input class, a lack of evidence can result in classes being incorrectly merged (or unmerged, conversely). As the input size grows, the amount of evidence required grows as well. As such, it is advisable to choose an input size that reflects the amount of evidence available. This may explain the poor performance of the model on the Xitsonga data set, as far fewer speech fragments were found for Xitsonga than for English.

Conclusion

A linear model for approximate partitioning of posteriorgrams was introduced. Using posteriorgrams from a Gaussian mixture model trained on MFCCs as a proof of concept, the model was shown to improve the across-speaker performance, with competitive results for the English data set. While the better-performing versions of the model depends on two hyperparameters, the hyperparameter search is alleviated somewhat by ease of training the linear model. Additionally, the entropy penalty was shown to be easy to optimise for, allowing a small value for the corresponding hyperparameter. The silhouette cluster separation measure was shown to be indicative of ABX performance, enabling hyperparameter search without making use of the gold transcription.

The resulting model is sparse and easily interpretable. However, the Jensen-Shannon loss function used is sensitive to the balancing of the same-class and different-class losses, making it particularly unsuitable for deep architectures.

Future work

A natural extension of this work is to use different probabilistic models to generate the posteriorgrams, and see how this affects the performance of the model. For instance, would the model be able to improve on the posteriorgrams generated by the model of Chen et al. (2015)? Of interest are also models that directly model time dependencies, such as hidden Markov models.

The model as presented here can be seen as a kind of radial basis function (RBF) network, where the RBF units (i.e. the Gaussian mixture model) are trained on the complete data set, while the output weights are trained using gradient descent on the fragment pair data. As such it might be interesting to see whether joint training of both the input clusters and the linear mapping by treating the model as a single RBF network would lead to any improvements.

Finally, as we have seen the Jensen-Shannon loss needs to reweighted in order to properly balance the same-class and different-class losses. It is thus desirable to find an alternative loss function suitable for probability distributions, for which the losses are naturally more balanced.

Bibliography

Barnard, E., Davel, M. H., Heerden, C. J. van, De Wet, F., and Badenhorst, J. (2014). "The NCHLT speech corpus of the South African languages." In: *SLTU* 2014, pp. 194–200 (cit. on p. 23).

Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., and Shah, R. (1994). "Signature Verification using a "Siamese" Time Delay Neural Network". In: *Advances in Neural Information Processing Systems*, pp. 737–744 (cit. on p. 18).

Chen, H., Leung, C.-C., Xie, L., Ma, B., and Li, H. (2015). "Parallel Inference of Dirichlet Process Gaussian Mixture Models for Unsupervised Acoustic Modeling: A Feasibility Study". In: *Sixteenth Annual Conference of the International Speech Communication Association* (cit. on pp. 1, 17, 19, 30).

Cooley, J. W. and Tukey, J. W. (1965). "An algorithm for the machine calculation of complex Fourier series". In: *Mathematics of computation* 19.90, pp. 297–301 (cit. on p. 4).

Endres, D. M. and Schindelin, J. E. (2003). "A new metric for probability distributions". In: *IEEE Transactions on Information theory* 49.7, pp. 1858–1860 (cit. on p. 21).

Glorot, X., Bordes, A., and Bengio, Y. (2011). "Deep Sparse Rectifier Neural Networks." In: *Aistats*. Vol. 15. 106, p. 275 (cit. on p. 15).

Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013). "Maxout networks". In: *ICML (3)* 28, pp. 1319–1327 (cit. on p. 15).

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press (cit. on p. 11).

Heck, M., Sakti, S., and Nakamura, S. (2016). "Unsupervised Linear Discriminant Analysis for Supporting DPGMM Clustering in the Zero Resource Scenario". In: *Procedia Computer Science* 81, pp. 73–79 (cit. on p. 18).

Hornik, K., Stinchcombe, M., and White, H. (1989). "Multilayer feedforward networks are universal approximators". In: *Neural networks* 2.5, pp. 359–366 (cit. on p. 14).

Huang, X., Acero, A., and Hon, H.-W. (2001). *Spoken language processing: A guide to theory, algorithm, and system development*. Prentice Hall (cit. on p. 3).

Jansen, A. and Church, K. (2011). "Towards Unsupervised Training of Speaker Independent Acoustic Models." In: *INTERSPEECH*, pp. 1693–1692 (cit. on p. 18).

Jansen, A., Thomas, S., and Hermansky, H. (2013). "Weak top-down constraints for unsupervised acoustic model training." In: *ICASSP*, pp. 8091–8095 (cit. on pp. 1, 18, 19).

Jansen, A. and Van Durme, B. (2011). "Efficient spoken term discovery using randomized algorithms". In: *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*. IEEE, pp. 401–406 (cit. on pp. 18, 23).

Kamper, H., Elsner, M., Jansen, A., and Goldwater, S. (2015). "Unsupervised neural network based feature extraction using weak top-down constraints". In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, pp. 5818–5822 (cit. on p. 19).

Kingma, D. and Ba, J. (2014). “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (cit. on p. 24).

Lam, S. K., Pitrou, A., and Seibert, S. (2015). “Numba: A llvm-based python jit compiler”. In: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. ACM, p. 7 (cit. on p. 24).

Lee, C.-y. and Glass, J. (2012). “A nonparametric Bayesian approach to acoustic model discovery”. In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*. Association for Computational Linguistics, pp. 40–49 (cit. on p. 17).

McFee, B., McVicar, M., Nieto, O., Balke, S., Thome, C., Liang, D., Battenberg, E., Moore, J., Bittner, R., Yamamoto, R., Ellis, D., Stoter, F.-R., Repetto, D., Waloschek, S., Carr, C., Kranzler, S., Choi, K., Viktorin, P., Santos, J. F., Holovaty, A., Pimenta, W., and Lee, H. (2017). *librosa 0.5.0*. DOI: 10.5281/zenodo.293021. URL: https://github.com/librosa/librosa (cit. on p. 24).

Montúfar, G. F., Pascanu, R., Cho, K., and Bengio, Y. (2014). “On the number of linear regions of deep neural networks”. In: *Advances in neural information processing systems*, pp. 2924–2932 (cit. on p. 14).

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press (cit. on pp. 9, 10).

Park, A. S. and Glass, J. R. (2008). “Unsupervised pattern discovery in speech”. In: *Audio, Speech, and Language Processing, IEEE Transactions on* 16.1, pp. 186–197 (cit. on p. 18).

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12, pp. 2825–2830. URL: http://scikit-learn.org/ (cit. on pp. 23, 24).

Pitt, M., Dilley, L., Johnson, K., Kiesling, S., Raymond, W., Hume, E., and Fosler-Lussier, E. (2007). *Buckeye Corpus of Conversational Speech (2nd release)*. Online. Columbus, OH: Department of Psychology, Ohio State University (Distributor). URL: www.buckeyecorpus.osu.edu (cit. on p. 23).

Quatieri, T. F. (2002). *Discrete-time speech signal processing: principles and practice*. Prentice Hall (cit. on p. 3).

Renshaw, D., Kamper, H., Jansen, A., and Goldwater, S. (2015). “A comparison of neural network methods for unsupervised representation learning on the Zero Resource Speech Challenge”. In: *Proc. Interspeech* (cit. on p. 19).

Al-Rfou, R. et al. (2016). “Theano: A Python framework for fast computation of mathematical expressions”. In: *arXiv e-prints* abs/1605.02688. URL: http://arxiv.org/abs/1605.02688 (cit. on p. 24).

Rousseeuw, P. J. (1987). “Silhouettes: a graphical aid to the interpretation and validation of cluster analysis”. In: *Journal of computational and applied mathematics* 20, pp. 53–65 (cit. on p. 26).

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). “Learning representations by back-propagating errors”. In: *Nature* 323, pp. 533–536 (cit. on p. 16).

Schatz, T., Peddinti, V., Bach, F., Jansen, A., Hermansky, H., and Dupoux, E. (2013). “Evaluating speech features with the Minimal-Pair ABX task: Analysis of the classical MFC/PLP pipeline”. In: *INTERSPEECH 2013: 14th Annual Conference of the International Speech Communication Association*, pp. 1–5 (cit. on p. 27).

Siu, M.-h., Gish, H., Chan, A., Belfield, W., and Lowe, S. (2014). “Unsupervised training of an HMM-based self-organizing unit recognizer with applications to topic classification and keyword discovery”. In: *Computer Speech & Language* 28.1, pp. 210–223 (cit. on p. 17).

Stevens, S. S., Volkmann, J., and Newman, E. B. (1937). “A scale for the measurement of the psychological magnitude pitch”. In: *The Journal of the Acoustical Society of America* 8.3, pp. 185–190 (cit. on p. 5).

Synnaeve, G. and Dupoux, E. (2016). “A Temporal Coherence Loss Function for Learning Unsupervised Acoustic Embeddings”. In: *Procedia Computer Science* 81, pp. 95–100 (cit. on p. 18).

Synnaeve, G., Schatz, T., and Dupoux, E. (2014). “Phonetics embedding learning with side information”. In: *Spoken Language Technology Workshop (SLT), 2014 IEEE*. IEEE, pp. 106–111 (cit. on pp. 1, 18, 27).

Thiolliere, R., Dunbar, E., Synnaeve, G., Versteegh, M., and Dupoux, E. (2015). “A hybrid dynamic time warping-deep neural network architecture for unsupervised acoustic modeling”. In: *Proc. Interspeech* (cit. on pp. 1, 18, 19, 27).

Varadarajan, B., Khudanpur, S., and Dupoux, E. (2008). “Unsupervised learning of acoustic sub-word units”. In: *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*. Association for Computational Linguistics, pp. 165–168 (cit. on p. 17).

Versteegh, M., Anguera, X., Jansen, A., and Dupoux, E. (2016). “The Zero Resource Speech Challenge 2015: Proposed Approaches and Results”. In: *Procedia Computer Science* 81, pp. 67–72 (cit. on p. 18).

Versteegh, M., Thiolliere, R., Schatz, T., Cao, X. N., Anguera, X., Jansen, A., and Dupoux, E. (2015). “The zero resource speech challenge 2015”. In: *Proc. of INTERSPEECH* (cit. on pp. 1, 18).

Young, S., Evermann, G., Gales, M., Hain, T., Kershaw, D., Moore, G., Odell, J., Ollason, D., Povey, D., Valtchev, V., and Woodland, P. (2005). *The HTK Book*. Cambridge University Engineering Department (cit. on p. 7).

Zeghidour, N., Synnaeve, G., Versteegh, M., and Dupoux, E. (2016). “A deep scattering spectrum — Deep Siamese network pipeline for unsupervised acoustic modeling”. In: *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, pp. 4965–4969 (cit. on p. 19).

Zhang, Y. and Glass, J. R. (2010). “Towards multi-speaker unsupervised speech pattern discovery”. In: *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*. IEEE, pp. 4366–4369 (cit. on p. 18).