Contents

List of Figures

Chapter 1

Mathematical Background

1.1 Introduction

An important part of every pattern recognition project handling large amounts of data, is the dimensionality reduction problem, that is, the transfer of the input data into a lower dimensional space, which will allow for more efficient processing and lower model error, as fewer parameters will have to be estimated. In order to develop algorithms for this task, one must be able to understand basic notions often arising in such context, e.g. Hilbert space, convergence, metric, manifolds. The purpose of this chapter is to offer a brief introduction to these concepts in connection with the broader topic of the project([Kre78]).

1.2 Metric Spaces

A *space* in the broader mathematical sense, is a set of elements X with some added structure.

Specifically, we will be working on *metric spaces*. Prior to defining a *metric space* we must first define what a *metric* is.

Definition 1. *A metric d on a set X (or distance function on X) is a function defined on* $X \times X$*, such that for all* $x,y,z \in X$ *,the following properties hold true:*

- *d is real-valued, finite and non-negative*
- $d(x,y) = 0$ *iff* $x = y$
- $d(x,y) = d(y,x)$ *(Symmetry)*
- $d(x,y) \leq d(x,z) + d(z,y)$ *(Triangle Inequality)*

Definition 2. *A metric space is a pair (X,d) where X is a set and d is a metric on X.*

Set X is also called the *underlying set* of (X,d).For the fixed points x,y the non-negative number *d(x,y)* is called the distance from x to y.

Examples of metric spaces are the well known $\mathbb{R}$ with the distance function

$$d(x, y) = |x - y|$$

, and the Euclidean space $\mathbb{R}^2$ with the Euclidean metric, defined by

$$d(x, y) = \sqrt{(x_1 - y_1) + (x_2 - y_2)}$$

for elements $x = (x_1, x_2)$ and $y = (y_1, y_2)$.

Spaces and their structure play a vital role in dimensionality reduction.The goal of this process is to transfer the input space into a lower dimensional one, yet at the same time retain the original relations between input data. In order to do this, we need to make sure that the space we move to, has certain properties.We are especially interested in spaces where the triangular inequality is satisfied, so that we can take advantage of the convergence properties it offers to the space.This is the reason why we built on a metric space, adding properties and operations, to derive new spaces which offer the required structure.

We will now introduce some auxiliary concepts which will enable a smooth transition into further defining kinds of spaces.

Definition 3. *Given a point $x_0 \in X$ and a real number $r > 0$, we define three types of sets:*

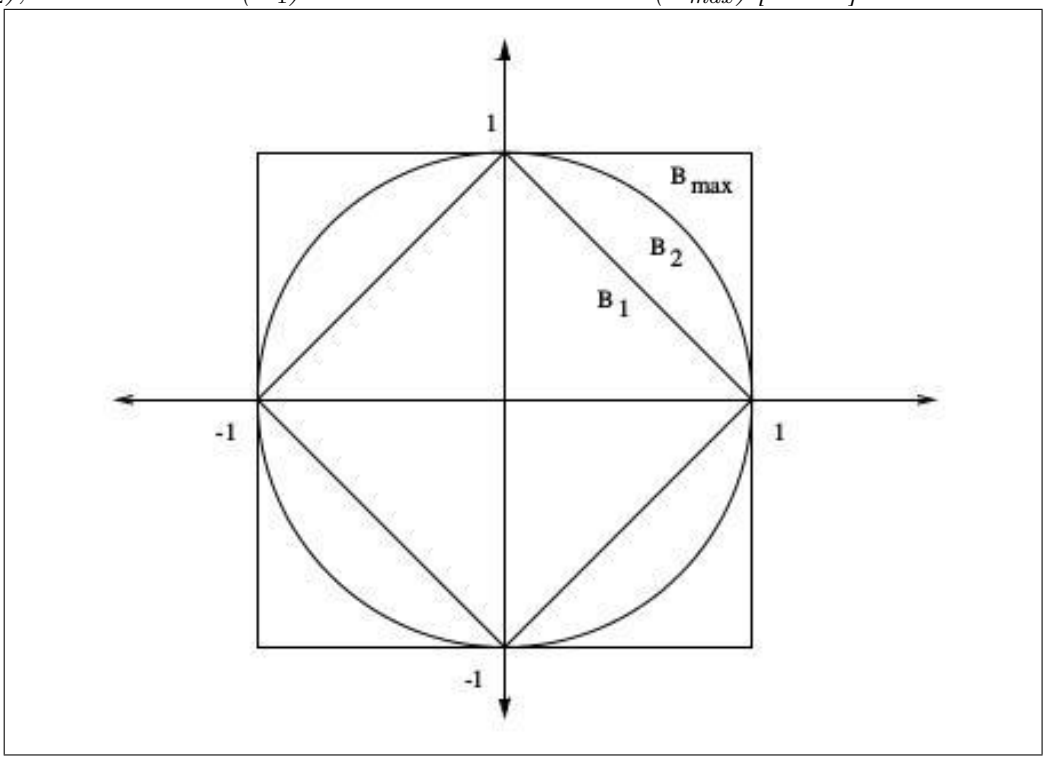
- *$B(x_0;r) = \{x \in X \mid d(x,x_0) < r\}$ (Open Ball)*
- *$B(x_0;r) = \{x \in X \mid d(x,x_0) \leq r\}$ (Closed Ball)*
- *$S(x_0;r) = \{x \in X \mid d(x,x_0) = r\}$ (Sphere)*

where point x_0 is called the center and r the radius.

Given the definition of a *ball*, we can define the notion of a neighborhood of a point.

Definition 4. *An open ball $B(x_0, \varepsilon)$, $\varepsilon > 0$, is called an ε - neighborhood of x_0. A neighborhood of x_0 is any subset of X which contains an ε - neighborhood of x_0.*

Figure 1.1: *The unit balls in the euclidean space defined using the euclidean norm (B_2), the sum norm (B_1) and the maximum norm (B_{max}).[Kre78]*



Since the process of dimensionality reduction is ,in its essence, a mapping from one space to another, we proceed by defining the notion of a continuous mapping.

Definition 5. *Let X = (X,d) and Y = (Y,$\bar{d}$) be metric spaces.A mapping $T : X \to Y$ is said to be continuous at a point $x_0 \in X$ if for every $\varepsilon > 0$ there exists a $\delta > 0$ such that*

$$\bar{d}(Tx, Tx_0) \ < \ \varepsilon \ \forall \ x \ satisfying \ d(x, x_0) \ < \ \delta$$

T is said to be continuous if it is continuous at every point of X.

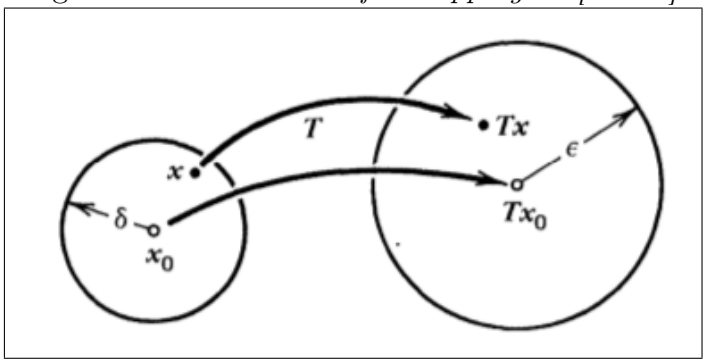
Having presented the ideas of a metric space and some of its subsets, as well as defined a continuous mapping, we proceed to remind the notions of convergence and completeness which will be used to move on to further spaces.

1.3 Convergence and completeness

Definition 6. *A sequence (x_n) in a metric space X = (X,d) is said to converge or to be convergent if there is a $x \in X$ such that*

$$\lim_{n\to\infty} d(x_n, x) = 0.$$

Figure 1.2: *Illustration of a mapping T. [Kre78]*



x is called the limit of x_n and we write

$$\lim_{n\to\infty} x_n = x \text{ or } x_n \to x.$$

It is obvious now why we need to be in a metric space to define the convergence of x_n: the metric d produces the sequence $a_n = d(x_n, x)$ whose convergence defines that of x_n.Furthermore, using the triangular inequality we can prove the following lemma:

Lemma 1. *Let X=(X,d) be a metric space.Then:*

- *A convergent sequence in X is bounded and its limit is unique.*
- *If $x_n \to x$ and $y_n \to y \in X$, then d(x_n, y_n)→ d(x,y).*

Convergence plays an important role in the definition of completeness which will help us later to define Banach and Hilbert spaces (where we mostly work).

Definition 7. *Let x_n be a sequence in $\mathbb{R}$ or $\mathbb{C}$. x_n converges and we call it a Cauchy sequence if and only if it satisfies the Cauchy convergence criterion,that is, if and only if for every given $\varepsilon > 0$ there is a N=N(ε) such that*

$$|x_m - x_n| < \varepsilon \quad \forall m, n > N.$$

However, this is the case only for $\mathbb{R}$ and $\mathbb{C}$. Given that most pattern recognition tasks work on multidimensional spaces, we have to generalize this convergence property in such spaces.

Definition 8. *A sequence x_n in a metric space X = (X,d) is said to be Cauchy (or fundamental),if for every $\varepsilon > 0$ there is a N=N(ε) such that*

$$d(x_m, x_n) < \varepsilon \quad \forall m, n > N.$$

This generalization allows us to give the following definition of completeness:

Definition 9. *A metric space X = (X,d) is said to be complete if every Cauchy sequence in X converges (that is, it has a limit which is an element of X).*

It is important to notice that convergence is not a property of the sequence by itself, but it also depends on the metric space that the sequence lies: the limit of the sequence must be *in the space*.

(EXAMPLES NEEDED??)

1.4 Vector spaces and Banach spaces

As we explore the structure of spaces so that we reach a suitable one for our problem, we come across the notion of *vector spaces*.

Definition 10. *A vector space (or linear space) over a field K is a non-empty set X of elements x,y,... ,which are called vectors, together with two algebraic operations : vector addition and multiplication of vectors by scalars, that is by elements of K.*

Vector addition associates with every ordered pair (x,y) of vectors, a vector x+y, called the sum of x+y, in such a way that the following properties are satisfied:

$$x + y = y + x$$

$$x + (y + z) = (x + y) + z$$

Furthermore, there exists a vector θ, called the zero vector, and for every vector x there exists a vector -x, such that for all vectors we have:

$$x + \theta = x$$

$$x + (-x) = \theta$$

Vector multiplication by scalars associates with every vector x and scalar α a vector αx (or $x\alpha$) called the product of α and x in such a way that for all vectors x,y and scalars α, β the following hold:

$$\alpha(\beta x) = (\alpha\beta)x$$

$$1x = x$$

$$\alpha(x + y) = \alpha x + \alpha y$$

$$(\alpha + \beta)x = \alpha x + \beta x$$

A vector space X exactly as defined above, may or may not be a metric space. To make sure that a relation between the algebraic structure of X and the metric exists, and thus be able to combine algebraic and metric concepts, we have to define on X a metric d based on a *norm*.

Definition 11. *A norm on a vector space X (over* $\mathbb{R}$ *or* $\mathbb{C}$*) is a real-valued function on X whose value at a* $x \in X$ *is denoted by*

$$\|x\|$$

and which has the properties

$$\|x\| \geq 0$$

$$\|x\| = 0 \Rightarrow x = 0$$

$$\|\alpha x\| = |\alpha|\|x\|$$

$$\|x + y\| \leq \|x\| + \|y\|$$

where x,y $\in X$ *and* $\alpha \in K$.

A norm on X defines a metric d on X which is given by

$$d(x, y) = \|x - y\| \quad x, y \in X$$

and is called the metric induced by the norm.

A vector space X with a norm defined on it, is called a normed space and is denoted by X or $(X,\|\cdot\|)$.

If a normed space is complete in the metric defined by the norm, we call it a *Banach space*.

1.5 Inner product spaces and Hilbert spaces

In a vector space, vectors can be added and multiplicated giving the space its algebraic properties. By defining a norm on such a space, the concept of the length of a vector is generalized allowing us to define a metric and establish a relation between the algebraic and geometrical structure of the space.

To connect the above with pattern recognition, one should notice that when the problem is transferred to a lower dimensional space, it is often desirable to keep not only the length/magnitude relations between the data (as represented by the distance/norm for example) but also the angle between them.This reminds us of the inner product of a Euclidean space, which is what we will expand over normed spaces.

Definition 12. *An inner product on a vector space X is a mapping of X×X into the scalar field K of X; that is, with every pair of vectors x and y, there is associated a scalar which is written*

$$\langle x, y\rangle$$

and is called the inner product of x and y, such that for all vectors x,y,z and scalar α the following properties hold:

$$\langle x + y, z\rangle = \langle x, z\rangle + \langle y, z\rangle$$

$$\langle \alpha x, y\rangle = \alpha\langle x, y\rangle$$

$$\langle x, y\rangle = \overline{\langle y, x\rangle}$$

$$\langle x, x\rangle \geq 0$$

$$\langle x, x\rangle \iff x = 0$$

An inner product on X defines a norm on X given by

$$\|x\| = \sqrt{\langle x, x\rangle}$$

and a metric induced by the above norm given by

$$d(x, y) = \|x - y\| = \sqrt{\langle x - y, x - y\rangle}$$

A vector space X with an inner product defined on it, is called an *inner product space* (or *pre-Hilbert space*).An inner product space which is complete in the metric defined by the inner product, is called a *Hilbert space*.

It is apparent that an inner product space is a normed space and a Hilbert space is a Banach space.

An important theorem of Banach spaces, and thus Hilbert spaces, is the *Fixed point theorem* or *Contraction theorem*.Before we present the theorem and its useful proof we have to give the following definitions.

Definition 13. *A fixed point of a mapping T:X → X of a set X into itself, is a x ∈ X which is mapped onto itself (it is kept "fixed" by T), that is,*

$$Tx = x$$

the image Tx coincides with x.

Definition 14. *Let X = (X,d) be a metric space. A mapping T:X → X is called a contraction on X if there is a positive real number $\alpha < 1$ such that for all x,y ∈ X*

$$d(Tx, Ty) \leq \alpha d(x, y)$$

that is, the images of any points x,y are closer together that the points x,y.

The Banach fixed point theorem is an existence and uniqueness theorem for fixed points of certain mappings, and its proof gives a constructive procedure for getting closer and closer to the fixed point starting from an initial approximation.Thinking this idea in connection with pattern recognition one should think of the fixed point as the pattern to be found and the initial approximation as the observation available.

Theorem 1. ***Banach fixed point theorem.***

Consider a metric space X = (X,d) where X ≠ ∅. Suppose that X is complete and let T:X→ X be a contraction on X. Then T has precisely one fixed point.

Proof Idea: We construct a sequence (x_n) and show that it is Cauchy so that it converges in the complete space X.Then we prove that its limit X is a fixed point of T and T has no further fixed points.

We choose any $x_0 \in$ X and define the iterative sequence (x_n) by

$$x_0\ ,\ x_1 = Tx_0\ ,\ x_2 = Tx_1 = T^2x_0\ ,...,\ x^n = T^nx_0,\ ... \tag{1.1}$$

which is the sequence of the images of x under repeated application of T. We now show that (x_n) is Cauchy. Since T is a contraction we have:

$$\begin{aligned} d(x_{m+1}, x_m) &= d(Tx_m, Tx_{m-1}) \\ &\leq \alpha d(x_m, x_{m-1}) = \alpha d(Tx_{m-1}, Tx_{m-2}) \\ &\leq \alpha^2 d(x_{m-1}, x_{m-2}) \\ ... &\leq \alpha^m d(x_1, x_0). \end{aligned}$$

Hence, using the triangular inequality we obtain for n>m:

$$\begin{aligned} d(x_m, x_n) &\leq d(x_m, x_{m+1}) + ... + d(x_{n-1}, x_n) \\ &\leq (\alpha^m + \alpha^{m+1} + ... + \alpha^{n-1}) d(x_0, x_1) \\ &= \alpha^m (\frac{1-\alpha^{n-m}}{1-\alpha}) d(x_1, x_0) \end{aligned}$$

Since 0< α <1, in the numerator we have 1 - $\alpha^{n-m} < 1$.Consequently,

$$d(x_m, x_n) \leq (\frac{\alpha^m}{1-\alpha}) d(x_1, x_0)$$

On the right 0< α <1 and $d(x_0, x_1)$ is fixed, so that we can make the right-hand side as small as we please by taking m sufficiently large (and $n > m$).This proves that (x_m) is Cauchy. Since X is complete, (x_m) converges, say $x_m \to x$. We now have to show that this limit x is a fixed point of the mapping T.

From the triangle inequality and the definition of contraction we have:

$$\begin{aligned} d(x, Tx) &\leq d(x, x_m) + d(x_m, Tx) \\ &\leq d(x, x_m) + \alpha d(x_{m-1}, x) \end{aligned}$$

The sum on the second line can be made smaller than any preassigned $\varepsilon > 0$ because $x_m \to$ x.We draw the conclusion that *d(x,Tx)=0)* and consequently $x = Tx$, which means that x is a fixed point of T.

We now have to show that this fixed point is unique. Let there be a second fixed point $\bar{x}$.Then

$$d(x, \bar{x}) = d(Tx, T\bar{x}) \leq \alpha d(x, \bar{x})$$

which implies that *d(x,$\bar{x}$)=0* since $\alpha < 1$. Hence x = $\bar{x}$ and the proof of the theorem ends here.

1.6 Manifolds

An important notion that we will come across very often in the present work is the notion of a *manifold*.

An intuitive representation of a manifold is the following: suppose we have a set of data vectors in $\mathbb{R}^n$, and that a subset of their dimensions represents certain features of interest.If we focus on these dimensions only,"keeping" the rest "steady", we notice that the data vectors move along a certain path in $\mathbb{R}^n$. This path, or curve formed in the space, is the *manifold* that the features lie on in $\mathbb{R}^n$.

For example, in figure 1.3 we see a curve in $\mathcal{R}^3$, which has zero volume and zero area. It can therefore be parameterized by a single variable. Consequently, despite being in the three-dimensional space, the curve has an intrinsic dimensionality of one, since it locally resembles $\mathcal{R}^1$. A manifold has a *locally* Euclidean geometry,in a neighborhood around each point, but on a global scale the relations between its elements are non metric.

We will give a formal definition of manifolds, when we talk about their use in representation learning.

1.7 Summary

In this chapter some basic ideas about spaces and their properties were presented.Starting from the metric and metric space, we moved on to the norm and vector space and based on these we defined the Hilbert space which is the closest generalization of the well-known Euclidean space. Hilbert spaces provide us with the tools we need in our work: convergence, completeness, contractions etc.

Figure 1.3: *A one-dimensional manifold embedded in the three-dimensional space. [Cay05]*

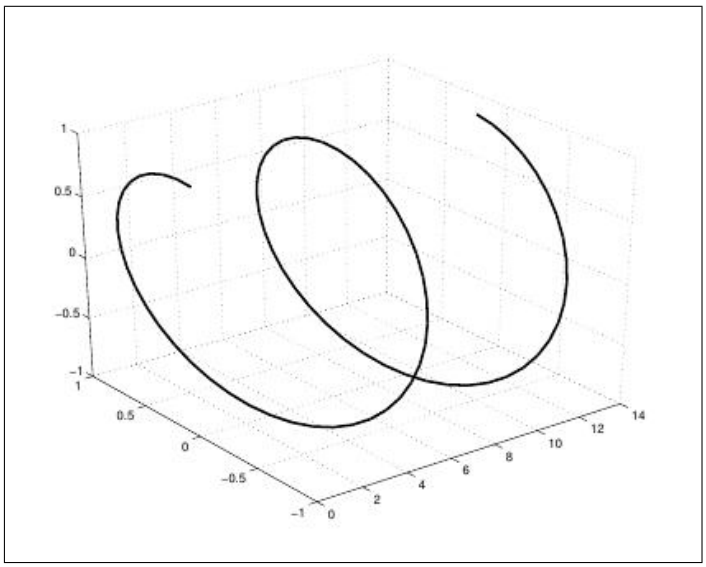
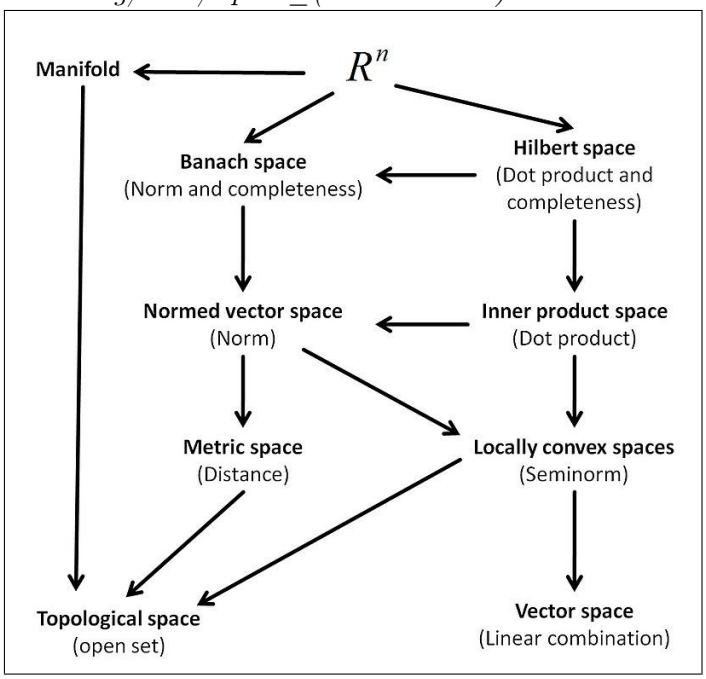


Figure 1.4: *Overview of various spaces and their relations.(Image taken from:https://en.wiki2.org/wiki/Space_(mathematics)*



Chapter 2

Automatic Speech Recognition

2.1 Introduction

The aim of the work on Automatic Speech Recognition (*ASR*) is to build systems that recognize spoken speech, that is, they are able to map acoustic signals to strings of words. In contrast to natural language processing, ASR does not try to determine the meaning behind speech or find the multiple meanings of words; it merely tries to recognize *which* words were spoken.

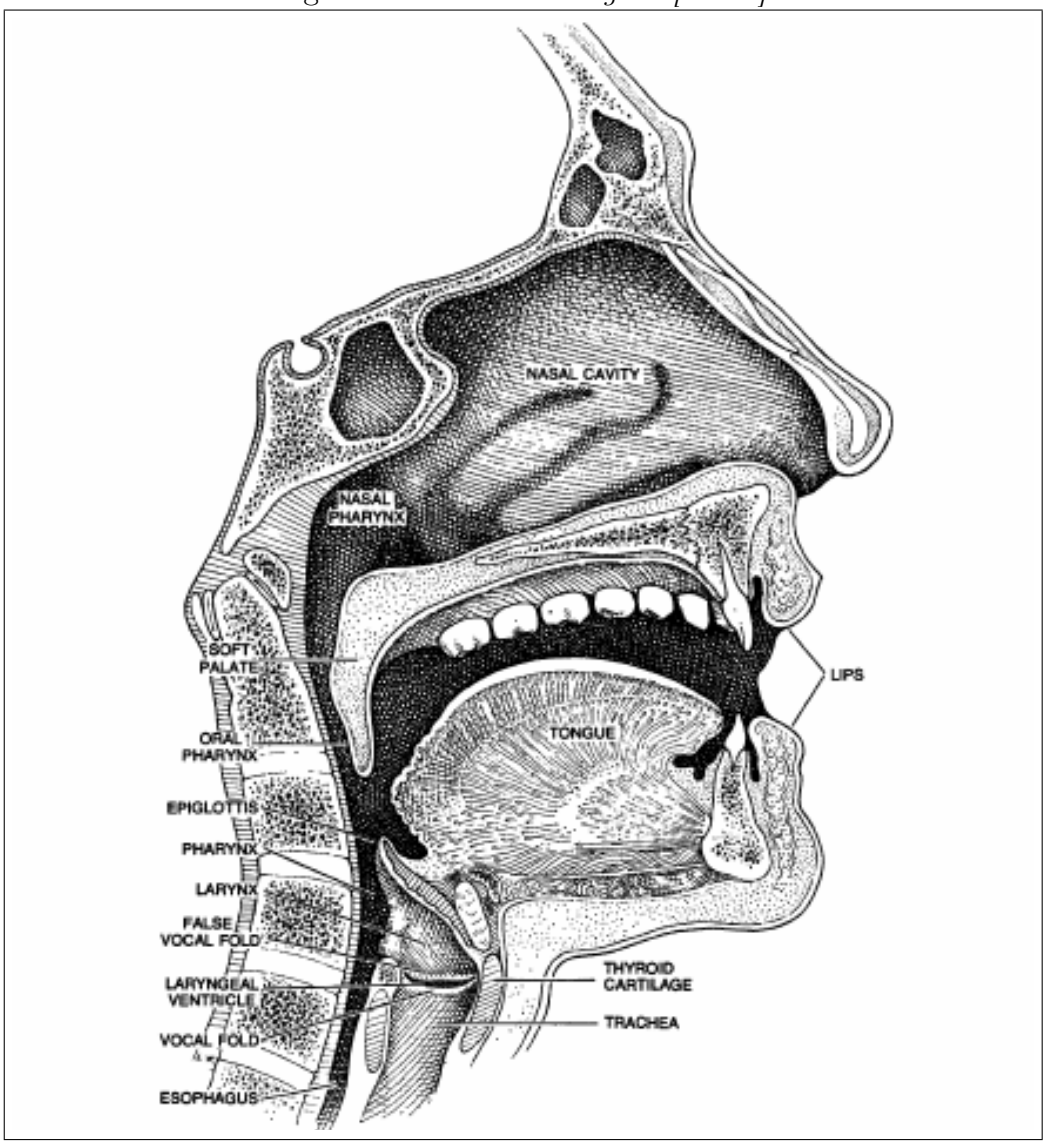
Research in ASR has come a long way in the last few years, which has allowed us to take advantage of it in multiple areas with very satisfying results: human-computer interaction (speech-only or multimodal interfaces), telephony (information passing, call routing) and dictation are examples where ASR systems can perform well irrespective of the speaker or their environment.

We will begin by providing a quick description of the way human speech is produced and move on to present the concept and mechanisms behind Automatic Speech Recognition.

2.2 Fundamentals of the speech production mechanism

[JM09] Sound is produced by the rapid movement of air from the lungs through the “windpipe", also called the *trachea* and out of the mouth or nose. As it flows through the trachea, it passes through the *larynx* (or *Adam's apple*) and there it affects the position of two small folds of muscle which are known as *vocal folds* or *vocal chords*. The possible movements of these muscles are either moving closer together or apart; the space between them is called the *glottis*. If the vocal folds are close together they will vibrate as air passes through the glottis and produce sounds

Figure 2.1: *The vocal organs.[JM09]*



that are known as *voiced* sounds. The vowels are examples of voiced sounds. On the other hand, if they are far apart they will not vibrate and the sounds produced are called *unvoiced*. Examples of unvoiced sounds are [p], [t] and [k].

After passing through the trachea and before exiting the body, air passes through the vocal tract, which consists of the *nasal* and *oral* tract. The vocal tract will act as a filter on the speech signal and the output of the filter will be the sound we will produce. The speech signal will vary according to what obstacles the air meets on its way out : the tongue, the lips or the teeth. These obstacles will define the vocal tract filter applied on the speech signal.

2.3 Design parameters of an ASR system

This section attempts to provide a short introduction into Automatic Speech Recognition and the most common architectures behind ASR systems[JM09].

However, before deciding on the architecture of a speech recognition system, one has to take into account its application domain.The most decisive parameters are:

- The vocabulary size,which is the number of distinct words that the system should be able to recognize. Few words imply relatively easy set-up and training of the system, whereas systems recognizing thousands of words, as in a broadcasting news vocabulary, are more complicated and harder to train.

- The fluency of the speech that the system will be asked to recognize.This includes whether the speech will be continuous or just isolated words as well as the speed and clarity of the speaker.Isolated word recognition systems, e.g. recognizing commands to a computer, are easier than ASR systems for continuous, conversational speech, e.g. a telephone conversation between humans.

- The environment where the recognition might have to take place. Systems designed to perform well in noisy environments with high distortion in the speech recorded are more demanding than systems that can clearly capture the speech for recognition in an isolated environment.

- The speaker variability. Speech recognition is easier if the system is expected to recognize the speech of a limited number of people. On the contrary, a general ASR system that should work with any speaker, regardless of sex, age, or accent is much more difficult to implement.

2.4 ASR system architecture

As mentioned in the introduction, the problem of ASR is, in principle, a structured sequence classification task, where a (relatively long) sequence of acoustic data is used to infer a (relatively short) sequence of the linguistic units such as words. Modern ASR systems use the model of a *noisy channel* to deal with the classification task. The idea behind this model is to think of the input signal as a distorted version of the corresponding words, which was produced as they passed through a noisy communications channel. If we manage to understand how the channel affected the signal, we can then match it to the original, noise-free set of words, by passing every acceptable -by the grammar of the language- sentence though the channel to get its distorted version and see which matches best the initial input signal.

Figure 2.2: *The noisy channel model. The decoder searches through all possible sentences and finds the one that after passing through the channel best matches the initial input signal[JM09].*

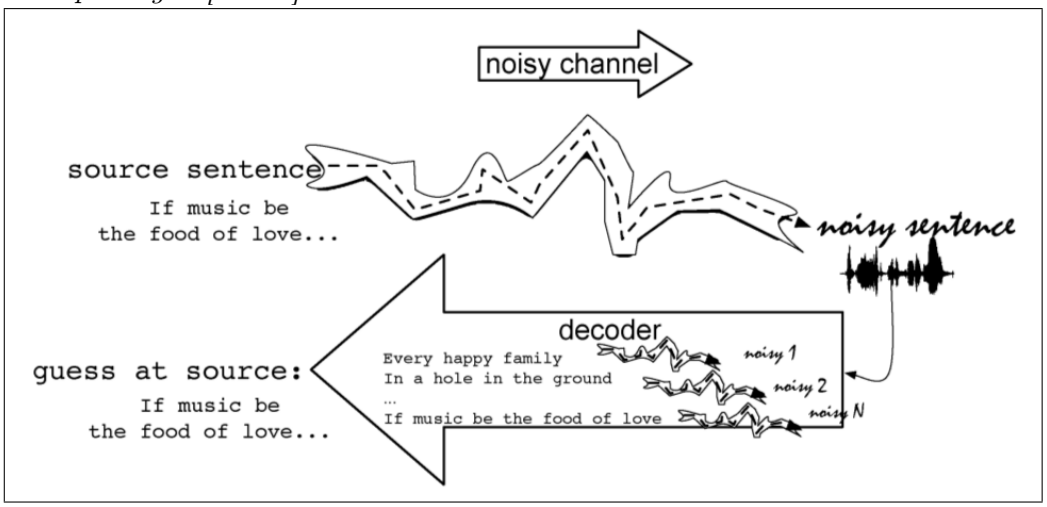
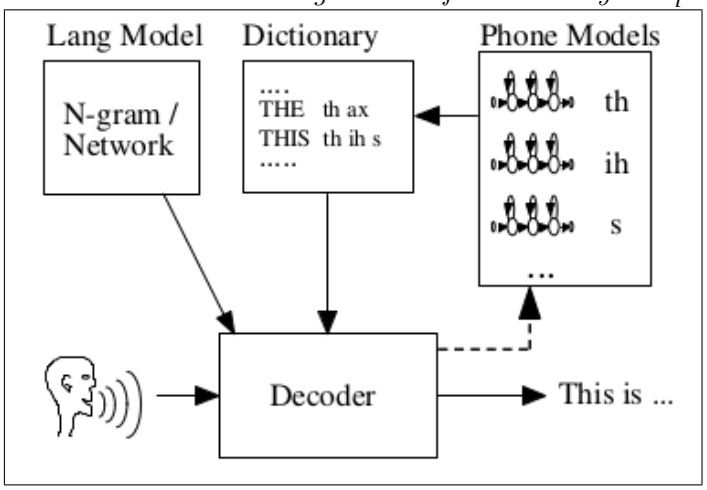


Figure 2.3: *The main building blocks of an ASR system[You02].*



In order to achieve the modelling of the noisy channel and the subsequent decoding of a new acoustic signal, we will need to have at our disposal the following tools: the prior probability of each sentence of the language, the probability of words being the concatenation of certain speech units and the probability of these speech units being realised as acoustic or spectral features, which are drawn from the input signal.

These tools define the main components of a modern automatic speech recognition system, which will be presented below, following the formulation of the computational/mathematical problem of Automatic Speech Recognition.

2.4.1 Computational formulation of ASR

We will now use mathematical notation and probability theory to answer the basic question in ASR: ''What is the most likely sentence Ŵ out of all sentences belonging to language $\mathbb{L}$ given an acoustic signal O? ''.

The acoustic input signal O is a sequence of observations o_i,

$$O = o_1, o_2, o_3, ...o_t$$

each one representing features of a specific part of the input speech, which is usually split into overlapping parts of a certain duration (*frames*). In the same way, if we treat each sentence of the language as a string of words,

$$W = w_1, w_2, w_3, ...w_n$$

we can write the answer to the ASR question in the following way:

$$\hat{W} = \underset{W \in \mathbb{L}}{\operatorname{argmax}} P(W|O)$$

, which we cannot compute directly. However, if we apply Bayes' rule, the equation takes the following form:

$$\hat{W} = \underset{W \in \mathbb{L}}{\operatorname{argmax}} \frac{P(O|W)P(W)}{P(O)}$$

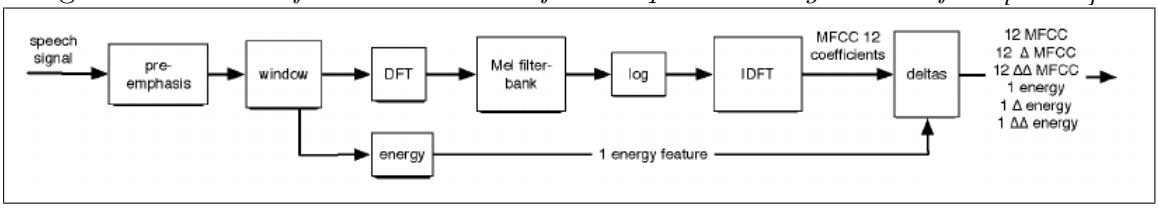
We can simplify the computation even further, if we consider that the probability of the observation in the denominator, *P(O)*, does not affect the maximization with respect to *W*, since the observation signal stays the same as we search over the sentences space. Consequently, the answer to our problem can be computed by the following form:

$$\hat{W} = \underset{W \in \mathbb{L}}{\operatorname{argmax}} P(O|W)P(W)$$

The two probabilities on the right hand side of the equation, represent the tools we need to address the recognition problem, as we have mentioned above: *P(W)*, which is the prior probability of each sentence of the language, is computed by the *language model*, whereas $P(O|W)$, which includes the probability of words being the concatenation of certain speech units and the probability of these speech units being realized as certain features, is calculated by the *acoustic model*.

Having presented the computational formulation of the automatic speech recognition problem, we can move on to present the required steps to be taken and models to be constructed in order to build a speech recognition system.

Figure 2.4: *MFCC feature extraction from a quantized digital waveform[JM09].*



2.4.2 Feature Extraction

[JM09] The first issue we have to address is how and in what format do we "insert" the speech waveform into our system. This process, known as *feature extraction* results in the extraction from the speech waveform of a sequence of acoustic *feature vectors*, each of them representing the information included in a small time window of the signal. These feature vectors are further preprocessed and finally presented as the input to the ASR system.

Mel Frequency Cepstral Coefficients

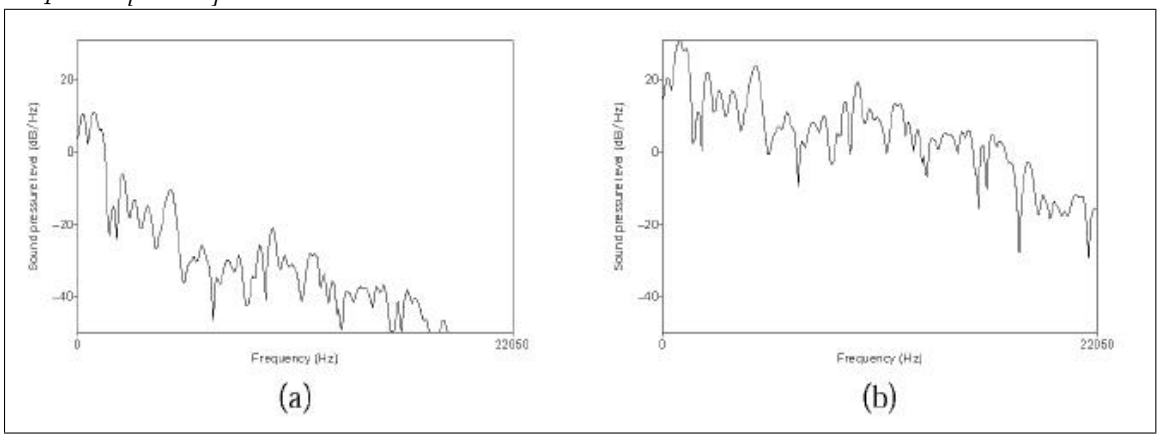
Although research in feature extraction moves towards using the raw waveform directly as input to the system, so far the most common feature representation in speech recognition systems is the Mel Frequency Cepstral Coefficients (*MFCC*).The steps involved in extracting the MFCC feature vectors follow the analog to digital conversion of the speech signal (sampling and quantization) and are outlined in the next paragraphs:

Pre-emphasis In the pre-emphasis step we want to amplify the amount of energy in the high frequencies of the signal. If we take a look at the spectrum of a vowel we will note a drop in energy as we move on to higher frequencies; this is known as *spectral tilt* and is due to the nature of the glottal pulse. **(EXPLAIN STH????)** Amplifying the energy of higher frequencies will improve phone detection accuracy as it will provide more information to the acoustic model coming from the boosted higher formants. In essence, pre-emphasis is applying to the signal a first order, high-pass filter whose equation is:

$$y[n] = x[n] - \alpha x[n-1],\ 0.9 \leq \alpha \leq 1.0$$

Windowing Given that the feature vectors we want to extract will be used to train phone classifiers, i.e. the acoustic model, we want them to able to capture the spectral properties corresponding to these fundamental speech units.Consequently, since speech is a non-stationary signal - its statistical properties are not constant

Figure 2.5: *Part of the spectrum of vowel [aa] before (a) and after (b) pre-emphasis[JM09].*



across time - we extract the feature vectors from a small window of the speech signal that corresponds to a phone or subphone, where the signal can be considered stationary - its statistical properties stay constant across time.

The windowing process comes down to applying a filter to the signal that is non-zero inside some region and zero elsewhere, moving this filter along the speech signal and extracting segments of the signal (or *frames*).The window is characterized by its *width* (in milliseconds, also called *frame size*), the *overlap* between successive windows (usually a percentage of the width) and its *shape*, e.g. rectangular, Hamming etc. The Hamming window is usually preferred over the rectangular as it gradually reduces the signal values at the boundaries of the window towards zero, thus avoiding discontinuities which cause problems during the next step of feature extraction (Fourier analysis):

$$w_{hamming}[n] = \begin{cases} 0.54 - 0.46\cos(\frac{2\pi n}{L}),\ 0 \leq n \leq L-1 \\ 0\ otherwise \end{cases}$$

The application of the filter is an element-wise multiplication of the signal values at each time step n with the values of the window:

$$y[n] = w[n]s[n]$$

where w is the window and s the signal.

Fourier Analysis The next step in the feature extraction process is to acquire the spectral information included in the windowed segments. The tool to extract such information, e.g. the amount of energy included in different discrete frequency bands of each segment, is the Discrete Fourier Transform.Given a part of a signal, the DFT will produce a complex number representing the magnitude and phase of

Figure 2.6: *Rectangular and Hamming windows and their effect on the signal[JM09].*

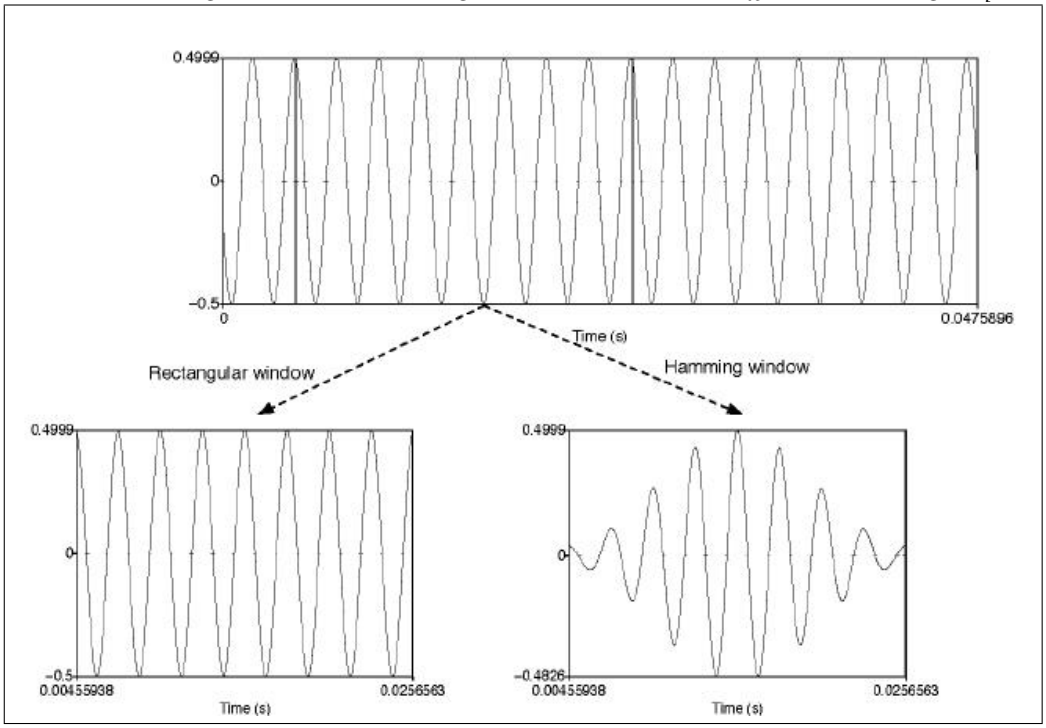


Figure 2.7: *Windowing process with a rectangular window.After a figure by Brian Pellom[JM09].*

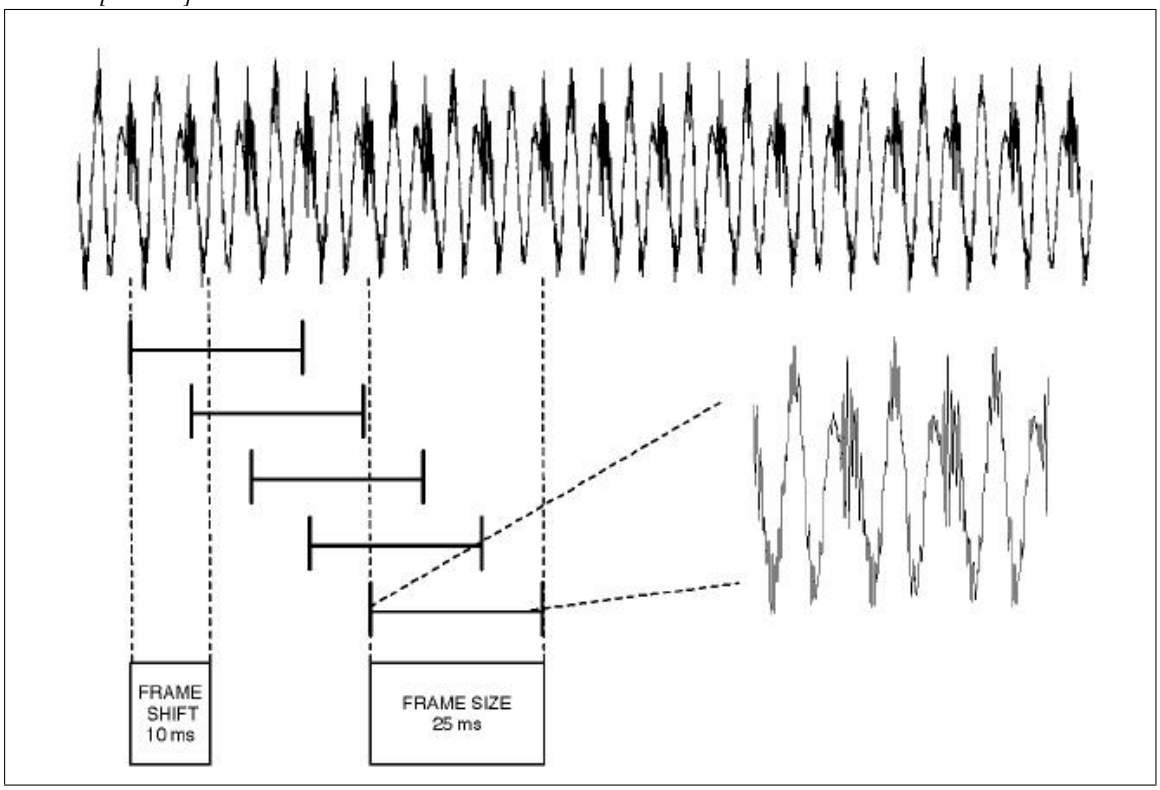
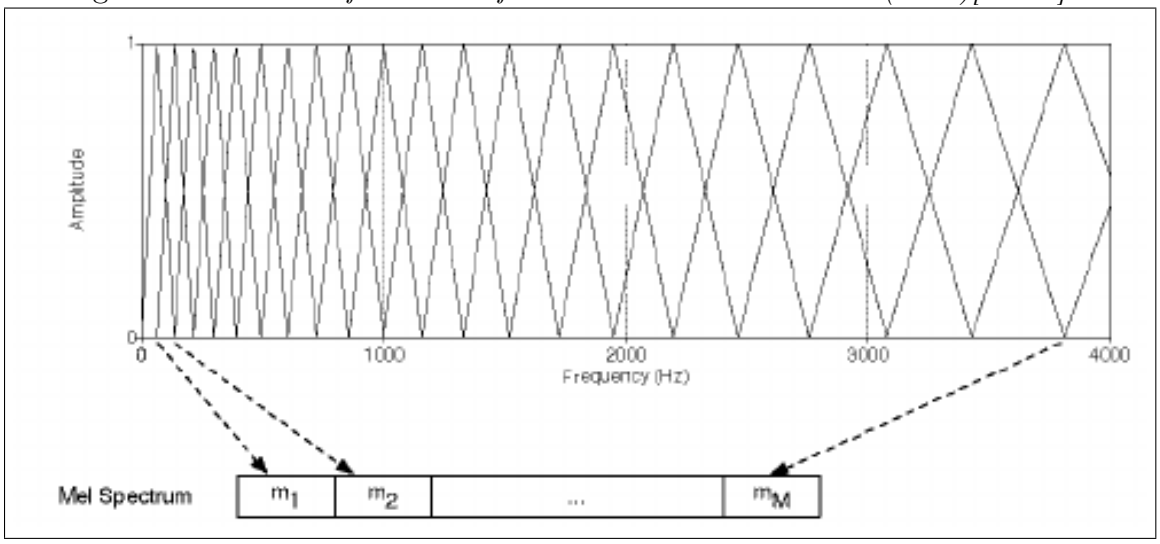


Figure 2.8: *The mel filterbank after Davis and Mermelstein (1980)[JM09].*



each frequency component of the corresponding segment.

Mel Filterbank Research has shown that the human hearing is not equally sensitive at all frequency bands - it is less sensitive at higher frequencies (above 1000 Hz).Moreover, humans are less sensitive to differences in amplitude at high amplitudes than at low amplitudes. Consequently, if we model this information and introduce it into the ASR system, we will improve its performance.

The way we take advantage of this information during the feature extraction process is by passing the DFT of the windowed signal parts through an array of triangular filters (a *filterbank*), which collect energy from each frequency band. These filters have their center frequencies spread on a *mel* scale, that is, 10 of them are spaced linearly below 1000 Hz and the remaining filters of the bank are spaced logarithmically above 1000 Hz. A *mel* is a unit of pitch defined so that pairs of sounds which are perceptually equidistant in pitch are separated by an equal number of mels[SVN37].The mapping between frequency in Hz and the mel scale is described by the relation:

$$mel(f) = 1127 \ln(1 + \frac{f}{700})$$

Finally, having passed the DFT of the signal through the filterbank, the final step is to take the logarithm of the mel spectrum values, which is a form of normalization to make the feature estimates less sensitive to variations in input.

The Cepstrum The next step in the MFCC feature extraction process is the computation of the *cepstrum*.As mentioned before, the speech waveform is created when a glottal source waveform is passed through the vocal tract which acts as a filter. As we have already mentioned, the shape of the vocal tract will determine the

Figure 2.9: *Effect of vocal tract on source signal.Tomi H. Kinnunen, Speech Technology Workshop*

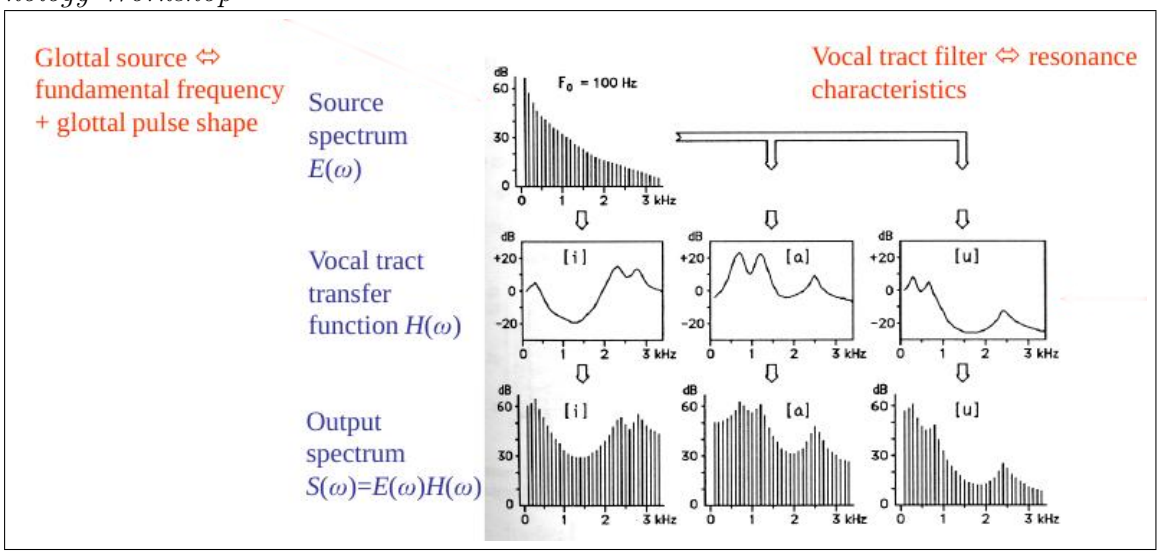
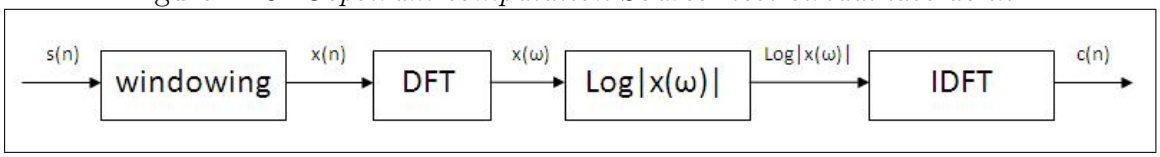


Figure 2.10: *Cepstrum computation.Source: test.virtual-labs.ac.in*

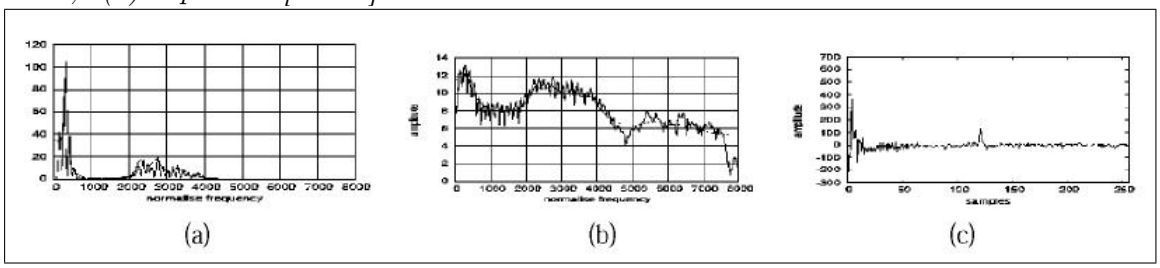


outcome of this filtering process, i.e. the sound, the phone that will be produced.

Therefore we aim to have some information about the vocal tract inserted into the feature vectors. The cepstrum provides us with a way of separating the glottal source from the vocal tract filter.

The peaks at lower values on the x-axis in the cepstrum of Figure 9 (c) correspond to the vocal tract characteristics whereas peaks at higher values correspond to the glottal source. Therefore, since we need information about the way a phone was produced, i.e. about the vocal tract shape, for the feature vectors we want to extract, we will keep a few of the first cepstral values (usually 12). Furthermore, an

Figure 2.11: *Cepstrum example: (a) magnitude spectrum, (b) log magnitude spectrum, (c) cepstrum.[JM09].*



important property of the cepstral coefficients is that their variance is uncorrelated, contrary to spectral coefficients, which are correlated at different frequency bands. This is extremely important for acoustic models based on Gaussian Mixture Models, as it allows us to keep the number of their parameters low.

Feature vectors The cepstral coefficients extracted from the previous process are just a part of the feature vectors. They are further enhanced by adding a few extra coefficients that provide more information helpful towards determining speech units such as phones.

The first extra piece of information we include by adding one more coefficient is the energy of the frame, which is defined as:

$$Energy = \sum_{t=t_1}^{t_2} s^2[t],$$

where s is the speech signal and t_1, t_2 are the boundaries of the frame.

Energy is useful for phone detection since it correlates with phone identity: higher energy reveals the presence of e.g. a vowel whereas very low energy could identify a pause in speech.

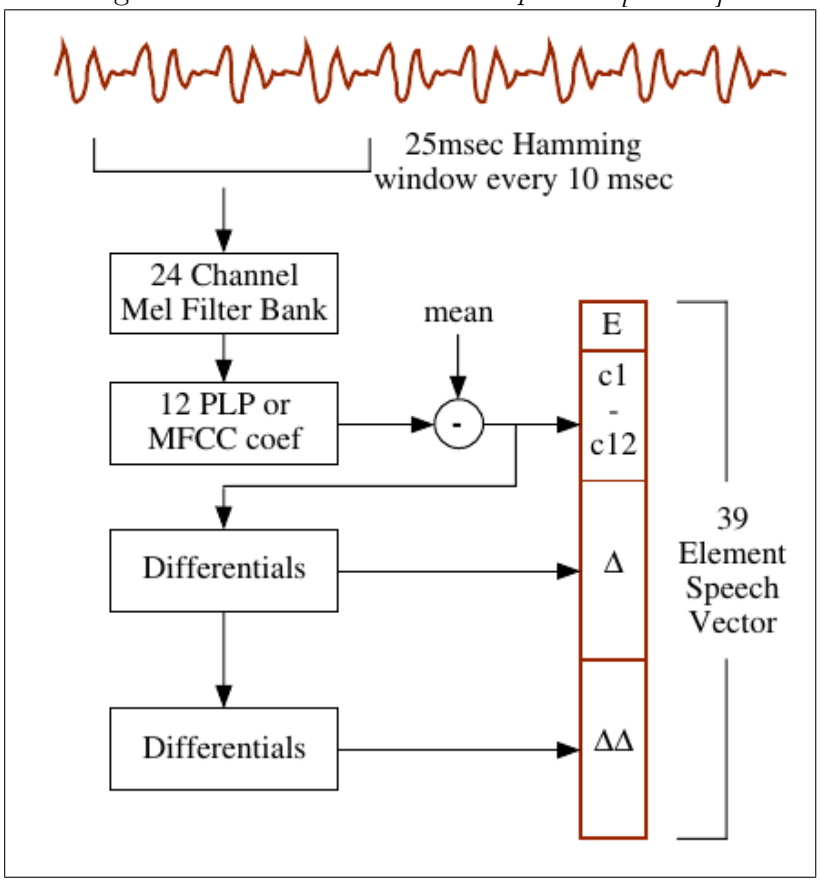
Considering that speech properties change from frame to frame it is reasonable to expect that capturing these changes would provide more information about the nature of the speech signal. This motivated the use of *delta* and *delta-delta* coefficients for each one of the cepstral coefficients and the energy. The *delta* coefficients capture the change of the corresponding feature between successive frames whereas the *delta-delta* coefficients capture the change of the *delta* features between successive frames. The simplest way to compute these *delta* and *delta-delta* coefficients is by taking the difference of the corresponding features between successive frames:

$$d(t) = \frac{c(t+1) - c(t-1)}{2},$$

where c is the cepstral feature with delta coefficient d at time t.

This concludes the construction of the feature vectors.As mentioned before, we usually pick 12 cepstral coefficients plus an energy coefficient, thus we will have 39-dimensional feature vectors: 13 + 13 *delta* +13 *delta-delta* coefficients.It is also common to concatenate the cepstral coefficients to produce higher dimensional vectors.This way we manage to include contextual information in the feature vectors.To deal with computational problems arising from the increase in the number of dimensions, various dimensionality reduction techniques are used, from simple ones such as Linear Discriminant Analysis or Principal Component Analysis, to more sophisticated such as techniques aiming to discover the lower dimensional manifold on which the feature vectors lie.

Figure 2.12: *Feature extraction process.[You02].*



2.4.3 The Language Model

The language model (LM) expresses how likely a given string of words is, taken into consideration certain linguistic constraints. In order to do this, we build on the idea of predicting the next word in a sequence of words, which is formalized with probabilistic models called *N-gram models*.

N-grams

An *N-gram* is a sequence of N words, e.g. a 2-gram or bigram is a sequence of two words, a 3-gram or trigram is a sequence of three words etc. An *N-gram model* is a probabilistic model which computes the N^{th} word of a sequence of N words given the previous N-1.

The power of N-grams becomes evident in areas such as speech or handwriting recognition, machine translation, spelling correction and natural language processing tasks. What all these areas have in common is that they might have to deal with noisy or ambiguous input. N-grams can deal with ambiguity by assigning a higher likelihood to word sequences that are valid according to the language constraints.

Since these models are capable of assigning a conditional probability to the next possible word in a group, we can exploit them to compute the joint probability of a sequence of words,i.e. a sentence, which is what we were aiming to from the beginning.

Suppose we have a sequence of n words $W = w_1, w_2, w_3, ..., w_n$. Then, the probability *P(W)* can be computed in the following way:

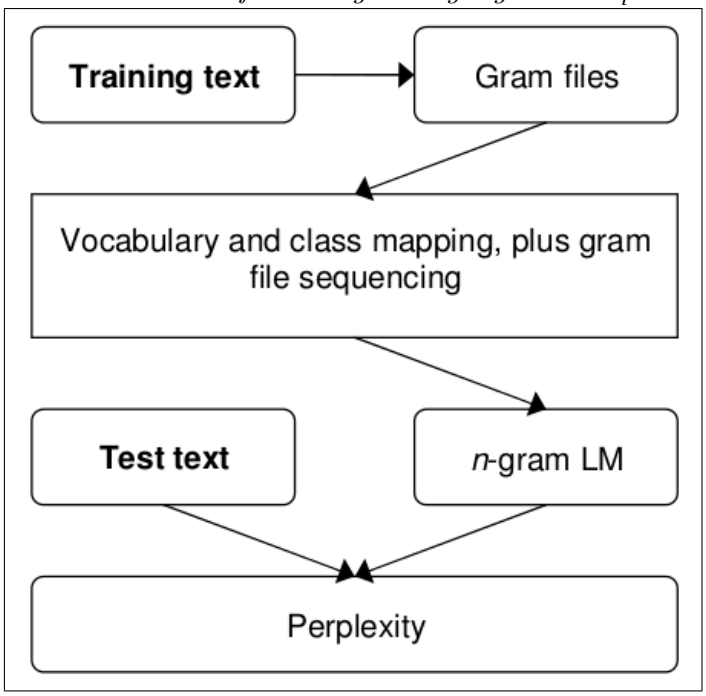
$$P(W) = P(w_1, w_2, w_3, ..., w_n) = \prod_{i=1}^{n} P(w_i|w_1, w_2, w_3, ..., w_{i-1})$$

As this computation for every possible word sequence in the language is very difficult if not impossible, we make the assumption that the i^{th} word depends only on the previous N-1 words (its *history*). At this point we can take advantage of the intuition behind the N-gram model:

$$P(W) = \prod_{i=1}^{n} P(w_i|w_1, w_2, w_3, ..., w_{i-1}) \approx \prod_{i=1}^{n} P(w_i|w_{i-N+1}, ..., w_{i-1})$$

to approximately compute the probability of the sequence. The expressions on the two sides would be exactly equal for sufficiently high n and if the language were ergodic, that is, the probability of any word could be estimated from sufficient history independent of the starting conditions.

Figure 2.13: *Process of building a language model.[YEK+02].*



Building a language model based on N-grams

[YEK+02]

Considering the intuition of N-grams and assuming that the probability of an N-gram occurring in an unknown text can be estimated from its frequency in a given training text, we can build language models based on N-grams.

The construction of such an LM can be broken down into three stages:

- Collect and store the N-grams of the training text (*corpus*)
- Possibly map some words into classes, e.g. out-of-vocabulary class mapping ???????????
- Count the N-grams and compute the N-gram probabilities

The last step of computing the probabilities is based on maximum likelihood estimation:

$$\hat{P}(w_i|w_{i-N+1},...,w_{i-1}) = \frac{C(w_{i-N+1},...,w_i)}{C(w_{i-N+1},...,w_{i-1})}$$

where *C(.)* is the count of a given word sequence extracted from the training text.

When building a language model based on N-grams, there are several factors one has to take into account before they decide on N. Resources constraints (e.g. storage) and size of the *vocabulary* (i.e. the set of distinct words in the language) will play a major role in deciding on N, as the number of parameters of the model grows

exponentially with N : $|V|^N$, where V is the vocabulary. However, storage needs because of parameters' size are lower than one would expect, because not all N-word combinations are acceptable/valid sequences in the language. What increases storage and computational needs however, is the large training sets required, so that our model estimates parameters with a minimum acceptable degree of confidence. Apart from processing, also acquiring these training sets might be difficult, especially for domain-specific applications, where training sets have to be specifically constructed.

Data sparsity and smoothing

[JM09] However, since the training set will always be finite, one can never have a sufficient number of N-grams for every valid N word sequence of the language. This issue faced when developing LMs is known as *data sparsity* and the technique used to deal with it is called *smoothing*. Smoothing aims to increase the robustness of the language model by redistributing the probability mass assigned by the maximum likelihood estimates: it removes some of it from higher counts of N-grams and assigns it to very low or zero counts, in order to ''smooth'' the distribution.

Laplace Smoothing. *Laplace* or *add-one* smoothing is the simplest form of smoothing : we just add one to all the counts. Before we compute the ML probabilities, one has to to take into account the extra $|V|$ ''words'' that we added, in order to maintain the sum of all probabilities equal to 1. As this last step is necessary and because Laplace smoothing does not perform well, it is more convenient to use an *adjusted count*

$$c_i^* = (c_i + 1)\frac{K}{K+V}$$

where c_i is the original count and K is the number of word tokens.

Discounting. Instead of adding the same amount of probability mass to all N-gram probabilities, a different approach would be to remove some mass out of the higher counts and assign it to lower ones. We can therefore define a *relative discount* as the ratio of the new counts and the originals:

$$d_c = \frac{c^*}{c}$$

One algorithm applying discounting, is the *Good-Turing discounting*.The intuition behind it is that we use the MLE of N-grams occurring c+1 times in the training set, to define the MLE of N-grams occurring c times. The new smooth count c^* is thus defined as:

$$c^* = (c+1)\frac{N_{c+1}}{N_c}$$

where N_i is the number of different N-grams occurring i times in the training set.

Back-off and interpolation. Discounting techniques allow us to distribute some probability mass equally to the unseen events. However, we can distribute it fairer if we take into consideration information from lower or higher order N-grams. This is the idea of *back-off* smoothing. In particular, *Katz back-off* always resorts to the (i-1)-gram if the i-gram has zero counts (i starting from our originally selected N). From a different point of view, we back-off to lower order N-grams only if we have zero evidence for a higher order one. On the other hand, *interpolation* deals with zero counts by summing estimates of all N-grams using weights (e.g. interpolate the estimates of unigrams,bigrams and trigrams).

Language model evaluation

Given the numerous applications of N-grams and LMs as well as their inherent drawbacks (static and finite vocabulary, finite training sets, more N-grams than can ever be collected and utilized), it is evident that we must have a way to compare LMs and evaluate their performance.

The most obvious way to compare two different language models would be to use them in our application and see which gives the best results. However, this way is expensive and time consuming as it is based on training and evaluating systems using huge speech datasets.

Another way to compare two LMs independent of the application, is to use the *perplexity* metric. The perplexity of an LM on a test set is a function of the probability that the LM assigns on it and is defined as:

$$PP(W) = P(w_1 w_2 w_3 ... w_N)^{-\frac{1}{N}}$$

where $W = w_1 w_2 w_3 ... w_N$ is the test set.

The best LM would be the one that has the minimum perplexity, since that would mean that it maximizes the test set probability, i.e. it better predicts the details of the test set. Another way to look at perplexity is as the weighted average branching factor of the language, that is, the number of best possible words following a word sequence. The smaller that number is, the better the work of the LM on coping with the ambiguity of the language.

Finally, when comparing language models using the perplexity metric, one should take care to use the *same vocabulary* for both LMs and evaluate them on the same test set which will be presented to the system for the first time during evaluation.

Recent advances in language modeling

In a paper presented recently [BDVJ03], a novel approach to building language models was presented which takes advantage of neural networks. The motivation

behind using NNs to build language models is based on the following problems associated with N-gram models:

- a huge amount of training data is needed to train LMs which will still have limited context capabilities (1-2 words)
- N-gram models ignore word similarity, which makes generalizing difficult

To deal with these issues becomes even more important when one considers that language models are probabilistic models using discrete random variables (words) which largely increases the amount of free parameters they need.

The writers suggest using distributed representations of words (i.e. real-valued feature vectors) which allows them 1) to identify similarities between words, since similar words will have similar feature vectors and 2) to exploit the smooth probability function modeled by a neural network in order to generalize: in this way, each vector representing a word will be able to provide information about a huge number of similar words, i.e. its "neighbors" in the feature space.In their work they present a neural network which simultaneously learns its parameters and the feature vectors associated with each word in the training set, and most importantly, the number of parameters it uses scales linearly with the vocabulary and context size. Due to the high computational cost of the training (higher than N-gram based models), they use parallel methods to efficiently train the model.

In recent years the use of neural networks for language modeling has included using recurrent neural networks which can take advantage of arbitrarily long contexts for each word (like humans do), something that was not possible with feedforward NNs [MKB+10].

2.4.4 The Acoustic Model

[You02]

According to the computational formulation of the ASR problem, we need the likelihood of the observed data (i.e. the acoustic signal) given the word sequence $P(O|W)$.However, it would be impractical and inefficient if we tried to compute this likelihood by building a separate model for each word in the language, since sub-word units are shared among different words. Instead, as mentioned before, the *acoustic model* calculates the probability of words being the concatenation of certain speech units and combines it with the probability of these speech units being realized as certain features, to produce the desired likelihood for a word.

We will first present some basic notions in acoustic modeling and then we will go into more details about this important part of the system.

Phones

The basic unit of speech analysis we use, is the *phone* which is the smallest identifiable unit we find in a stream of speech. The sequence of phones that constitutes each word in the training dataset is determined by a *pronouncing dictionary*. Using a phone sequence to represent each word makes it easy to add new words in the dataset just by adding them and their phone sequence to the dictionary.

Context-dependent phones

Given that there are thousands of words in a language, but just tens of phones (e.g. 44 in the English language) the computational and storage gain acquired from the use of phones as the basis of the acoustic model becomes immediately apparent. However, contextual effects like co-articulation cause large variations in the way that different sounds are produced even if in principle they correspond to the same phone. Hence, to achieve good phonetic discrimination, we build *context-dependent* phone models, with the most common being the *triphone*: for each phone there is a different model for every unique pair of left and right neighboring phones.

There are two dominant triphone models:

- *cross-word triphones*, which include phones of the previous and following words in the first and last triphones of the word of interest:

$$ten\ pots \rightarrow sil\ {}_{sil}t_e\ {}_te_n\ {}_en_p\ {}_np_o\ {}_po_t\ {}_ot_s\ {}_ts_{sil}\ sil$$

  The advantage of this approach is that they model co-articulation across word boundaries, but on the other hand, they complicate the decoding process since the phone models of each word depend on the following and preceding words as well.

- *word-internal triphones*, which explicitly encode word boundaries, thus making decoding easier:

$$ten\ pots \rightarrow sil\ {}_{sil}t_e\ {}_te_n\ {}_en_-\ {}_-p_o\ {}_po_t\ {}_ot_s\ {}_ts_{sil}\ sil$$

State of the art systems use mostly cross word triphones because of their ability to model contextual effects.

As a consequence, the number of distinct triphones greatly increases and the number of parameters for such systems can grow up to hundreds of millions, while at the same time we have too little training data in our disposal. In addition, we might

have unseen triphones appearing in evaluation tasks. To deal with these problems we have developed smoothing techniques, just as was the case with language models.

Smoothing techniques

Back-off and interpolation. When too little data is available for the training of a context-dependent model of a particular order, one can instead use a model of lower order at the expense of some inaccuracy in the modeling of the context: e.g. use a biphone or a context-independent phone (monophone) when we cannot use a triphone. In order to implement a more robust model one can use a weighted combination of models with various levels of context dependency (*interpolation*).

Parameter tying. An alternative that offers a greater degree of flexibility while maintaining the high level of context-dependency in the model, is the technique of *parameter-tying*, in which parameters of context-dependent phone models that are acoustically indistinguishable are tied together, to facilitate training in case there is little training data available.Before tying parameters together one has to apply some form of clustering to build the groups of phone models that will share their parameters.In practice, the most commonly used clustering technique is the *phonetic desicion tree*, where a binary tree is build for each phone model and its leaves contain the parameters to be shared.

Acoustic Modeling with Gaussian Mixture and Hidden Markov models

State-of-the-art ASR systems use Hidden Markov Models to represent each phone in conjunction with Gaussian Mixture Models to determine the probability that an acoustic observation was produced by a certain phone. The high representational capabilities and ease of training of these models are what has made them prevalent in ASR. We will present the GMM/HMM acoustic model and in the next chapters we will examine their most recent competitor, that is , deep neural networks.

Gaussian Mixture Models.[You02][YD14]

Multivariate Gaussian random variables and Mixture Models.

Given that the feature vector corresponding to an acoustic observation is multi-dimensional (usually 39-dimensional as was presented above), we will have to treat it as a multivariate random variable and use a multivariate distribution to assign a probability to it.The reason why we choose the Gaussian distribution is not only its desirable computational properties but also its ability to approximate many real-world data (owing to the law of large numbers), such as speech features.

Supposing that $\mathbf{\Sigma}$ is the co-variance matrix, $\boldsymbol{\mu}$ is the mean vector and d is the

number of dimensions of the feature vector, the multivariate Gaussian distribution is defined as:

$$p(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2} \left|\boldsymbol{\Sigma}\right|^{1/2}} \exp(-1/2(\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\vec{x} - \boldsymbol{\mu}) = \mathcal{N}(x|\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

However, because of the inherent multimodality of the speech features, a single Gaussian distribution is insufficient to describe them. Therefore, we use a *mixture* of Gaussian distributions:

$$p(\boldsymbol{x}) = \sum_{i=1}^{M} c_i \mathcal{N}(x|\boldsymbol{\mu_i}, \boldsymbol{\Sigma_i})$$

where c_i are the positive mixture weights and $\sum_{i=1}^{M} c_i = 1$.Usually, the number of the mixture components is chosen according to the nature of the problem and the information we have about the data.The variability and multimodality of the features may be due to multiple latent causes; provided we can identify these underlying causes we can match each one to the corresponding mixture component in the distribution.

As mentioned earlier, the Gaussian distribution is favorable both for its modeling and its computational properties. GMMs can model complex, multimodal distributions to any required level of accuracy and they can be trained using standard maximum likelihood approaches.Their attractiveness is also due to research into GMM training having come up with approaches to optimize the trade-off between their modeling effectiveness and the amount of training time and data needed. For example, we have the ability to reduce the number of free parameters (from M $\times d^2$ down to M) while still achieving high performance, if instead of using full covariance matrices we opt for diagonal Σ or even use the same matrix for all mixture components. The use of diagonal co-variance matrices has been thought to impose uncorrelatedness among features, but, given that a mixture of Gaussians with diagonal Σ can at least effectively describe the correlations modeled by a single full co-variance Gaussian, this thought has been misleading.

Specifically for speech recognition, a number of ways has been proposed to improve recognition accuracy of a GMM system.We can discriminatively train the system after the generative maximum-likelihood training, so that we maximize the probability of generating the observed speech features in the training data, or augment the input speech features with bottleneck features acquired using neural networks (the latter will be examined later on in this project).

The set of free parameters to be estimated for a Gaussian-mixture distribution is denoted by $\boldsymbol{\Theta}$ and consists of : $\{c_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}$. In order to acquire the parameters

we rely on maximum likelihood methods and in particular on the Expectation-Maximization (EM) algorithm [DLR77]. The EM algorithm is used to find locally maximum likelihood parameter estimates of statistical models when the equations cannot be solved directly.It is especially useful for models involving latent variables apart from the parameters-to-be-estimated and the observable data. A GMM can be treated as such a model if we assume that each observable data point has a corresponding hidden data point specifying the component of the mixture that each point belongs to.Furthermore, the EM algorithm provides us with closed-form expressions for the computation of the estimates in the M-step:

$$c_i^{(j+1)} = \frac{1}{N}\sum_{t=1}^{N} h_i^{(j)}(t),$$

$$\boldsymbol{\mu}_i^{(j+1)} = \frac{\sum_{t=1}^{N} h_i^{(j)}(t)x^{(t)}}{\sum_{t=1}^{N} h_i^{(j)}(t)},$$

$$\boldsymbol{\Sigma}_i^{j+1} = \frac{\sum_{t=1}^{N} h_i^{(j)}(t)[\boldsymbol{x}^{(t)} - \boldsymbol{\mu}_i^{(j)}][\boldsymbol{x}^{(t)} - \boldsymbol{\mu}_i^{(j)}]^T}{\sum_{t=1}^{N} h_i^{(j)}(t)}$$

where the posterior probabilities, i.e. the "latent" variables corresponding to the mixture components, computed in the E-step are:

$$h_i^{(j)}(t) = \frac{c_i^{(j)}\mathcal{N}(\boldsymbol{x}^{(t)}|\boldsymbol{\mu}_i^{(j)}, \boldsymbol{\Sigma}_i^{(j)})}{\sum_{m=1}^{n} c_m^{(j)}\mathcal{N}(\boldsymbol{x}^{(t)}|\boldsymbol{\mu}_m^{(j)}, \boldsymbol{\Sigma}_m^{(j)})}$$

The last equation computes the conditional probability for a given data point $\boldsymbol{x}^{(t)}, t = 1, ..., N$ being generated from mixture component i using the current (denoted by j) parameter estimate.

Despite the ease of training of GMMs, they have two serious drawbacks when it comes to speech recognition systems.The first one is that they cannot model the sequence information contained in speech features. To balance their inability, we combine GMMs with more general models able to capture sequence information: the Hidden Markov Models, which will be presented next. The second disadvantage, is that, in spite of their huge modeling capabilities, GMMs are statistically inefficient for modeling data lying on or near a nonlinear manifold in the data space; this is the case however for speech features, despite their seemingly high dimensionality.To deal with this matter, there are a number of techniques we can apply to extract

the lower dimensional manifold of the features. We will go into more details about manifolds and speech features in the following chapter.

Hidden Markov Models and acoustic modeling.[YD14]

As we have already mentioned, mixture-of-Gaussian random variables (single- or multidimensional) lack a “temporal" dimension, which would make the length of the random vectors variable, in order to follow the length of the speech sequence we intent to model. Therefore, although Gaussian mixture models are appropriate for short-term sound patterns, we will need to introduce a new model appropriate for sequences of speech acoustic vectors.

Extending the notion of the random variable to the discrete-time random sequence will provide us with the necessary tool to model acoustic vector sequences of variable length. A discrete-time random sequence is a collection with variable length, consisting of random variables indexed by uniformly spaced discrete times.We will focus on the most commonly used class of random sequences which is the Markov sequences.

Markov sequences

The concept of *state* is a key point in Markov sequences. If we think of a system functioning as a Markov sequence generating random variables, then the configuration of the system at each time step is defined by a specific *state* of the sequence. If the state of the Markov sequence is confined to be discrete, then the Markov sequence is called a *Markov chain* and the possible values of each discrete state constitute the discrete state space. When each discrete state value is generalized to be a new random variable (either discrete or continuous) the Markov chain is generalized to the *Hidden Markov Sequence*, also called *Hidden Markov Model* when it characterizes statistical properties of real-world data sequences.The Hidden Markov Model is the tool that we will use to model speech units used, such as sub-phones.

Markov chains

A Markov chain is a discrete-time Markov sequence.Its state space is of discrete nature, finite and each element of the space is associated with a state in the chain:

$$q_t \in s^{(j)}, j \ = \ 1, 2, ..., N$$

where q_t symbols a state.

A Markov chain denoted by $q_1^T = q_1, q_2, ..., 1_T$, is completely characterized by the initial state distribution probabilities (*priors*) and the transition probabilities defined by:

$$P(q_t = s^{(j)} | q_{t-1} = s^{(i)}) \doteq a_{ij}(t), \quad i, j \ = \ 1, 2, ..., N$$

Given the transition probabilities of a Markov chain, the state-occupation probability

$$p_j \doteq P[q_t = s^{(j)}]$$

can be recursively computed by

$$p_i(t) = \sum_{j=1}^{N} a_{ij} p_j(t-1), \quad \forall i$$

Hidden Markov Models[RJ86][Rab89][DHS00]

If the states of a Markov chain are *emitting*, that is, they are able to generate observational output variables, then we call the chain an observable Markov sequence. However, since there is an one-to-one correspondence between the output of the chain and the states, the model is inadequate to describe real-world informational sources such as sequences of speech features.To overcome this limitation, we will add randomness to the Markov chain by associating each state with an observation probability distribution, thus creating the hidden Markov sequence. It is called hidden because the underlying Markov chain is no longer directly observable but it can be observed only through a separate random function characterized by the observation probability distributions, which overlap across the states.

A Hidden Markov Model (*HMM*) is characterized by:

- N, the number of states in the model
- K, the number of distinct observation symbols per state
- The transition probabilities, $\mathbf{A} = [\alpha_{ij}]$, i,j = 1,2,...,N where

  $$\alpha_{ij} = P(q_t = j | q_{t-1} = i), \; i, j \; = \; 1, 2, ..., N$$

- The initial Markov chain prior probabilities

  $$\pi \; = \; [\pi_i], \; i \; = \; 1, 2, ..., N$$

  where π_i = P(q_1=i)

- The observation probability distribution, P($\boldsymbol{o}_t | s^{(i)}$), i = 1,2,...,N.

  If $\boldsymbol{o}_t$ is discrete, the distribution associated with each state gives the probabilities of symbolic observation $\boldsymbol{v}_1, \boldsymbol{v}_2, ..., \boldsymbol{v}_k$:

  $$b_i(k) = P[\boldsymbol{o}_t = \boldsymbol{v}_k | q_t = i], \quad i \; = \; 1, 2, ..., N.$$

  If the observation probability distribution is continuous, then the parameters $\boldsymbol{\Theta}_i$ in the p.d.f. characterize state i in the HMM.The most common p.d.f used

Figure 2.14: *Generate observation sequence from an HMM .[YD14]*

procedure DRAWFROMHMM(A, π, $P(\mathbf{o}_t|s^{(i)})$)
▷ A is the transition probability
▷ π is the initial state occupation probability
▷ $P(\mathbf{o}_t|s^{(i)})$ is the observation probability given a state (either Eq. 3.7 if discrete or Eq. 3.8 if continuous
Select an initial state $q_1 = s^{(i)}$ by drawing from the discrete distribution π
for $t \leftarrow 1; t \leq T; t \leftarrow t+1$ **do**
Draw an observation $\mathbf{o}_t$ based on $P(\mathbf{o}_t|s^{(i)})$
Make a Markov-chain transition from the current state $q_t = s^{(i)}$ to a new state $q_{t+1} = s^{(j)}$ according to the transition probability a_{ij}, and assign $i \leftarrow j$.
end for
end procedure

in speech processing is, as we have seen, the multivariate mixture of Gaussian distributions:

$$b_i(\boldsymbol{o}_t) = \sum_{m=1}^{M} c_{i,m}\mathcal{N}(\boldsymbol{o}_t|\boldsymbol{\mu_{i,m}}, \boldsymbol{\Sigma_{i,m}})$$

with $\boldsymbol{\Theta}_i = \{c_{i,m}, \boldsymbol{\mu}_{i,m}, \boldsymbol{\Sigma}_{i,m}\}$

Given these parameters one could consider the HMM as a generative model producing a sequence of observational data, $\boldsymbol{o_t}$, $t = 1, 2, ...T$. According to this perspective, the data at each time t is generated from the model according to:

$$\boldsymbol{o_t} = \boldsymbol{\mu_i} + \boldsymbol{r_t}(\boldsymbol{\Sigma_i})$$

where state i at a given time t is determined by the evolution of the Markov chain characterized by α_{ij} and

$$\boldsymbol{r_t}(\boldsymbol{\Sigma_i}) = \mathcal{N}(0, \boldsymbol{\Sigma_i})$$

is a zero-mean, independent and identically distributed (*IID*) residual sequence.Given that $\boldsymbol{\mu_i}$ is constant, the observation $\boldsymbol{o_t}$ is also IID given the state. Consequently, the HMM would produce locally stationary sequences making it appropriate to model sub-phone units.A procedure to generate sequences of observations from an HMM is described in the figure 2.14.

The three basic problems for an HMM.

Given the HMM model as presented above, there are three main problems associated with it that apply to real-world problems:

- The evaluation problem: Suppose we have an HMM $(A_{ij}, b_{ik}, \Theta_i)$.How do we determine the probability that a given sequence of observations was generated by that model?

- The decoding problem: Suppose we have an HMM $(A_{ij}, b_{ik}, \Theta_i)$.How do we determine the most likely hidden state sequence that led to the generation of a given observation sequence?

- The parameter estimation problem: Given the basic structure of an HMM (number of states and number of distinct observation symbols) as well as a set of training observations, how do we determine the parameters $(A_{ij}, b_{ik}, \Theta_i)$?

The evaluation problem. Let $\boldsymbol{q}_1^T$ be a finite length sequence of states in a Gaussian-mixture HMM and $P(\boldsymbol{o}_1^T, \boldsymbol{q}_1^T)$ be the joint likelihood of the observation sequence $\boldsymbol{o}_1^T$ and the state sequence $\boldsymbol{q}_1^T$.

Then, $P(\boldsymbol{o}_1^T|\boldsymbol{q}_1^T)$ denotes the likelihood that the observation sequence $\boldsymbol{o}_1^T$ is generated by the model conditioned on the state sequence $\boldsymbol{q}_1^T$ and is in the form of:

$$\prod_{i=1}^{T} b_i(\boldsymbol{o}_t)$$

whereas the probability of state sequence $\boldsymbol{q}_1^T$ is the product of transition probabilities:

$$P(\boldsymbol{q}_1^T) = \pi_{q_1} \prod_{t=1}^{T-1} a_{q_t q_{t+1}}$$

The joint likelihood $P(\boldsymbol{o}_1^T, \boldsymbol{q}_1^T)$ can be obtained as:

$$P(\boldsymbol{o}_1^T, \boldsymbol{q}_1^T) = P(\boldsymbol{o}_1^T|\boldsymbol{q}_1^T)\boldsymbol{q}_1^T$$

Since the hidden state sequence $\boldsymbol{q}_1^T$ is not known, we will have to sum over all possible state sequences in order to compute the desired probability:

$$P(\boldsymbol{o}_1^T) = \sum_{\boldsymbol{q}_1^T} P(\boldsymbol{o}_1^T, \boldsymbol{q}_1^T)$$

The amount of this computation though, is exponential in the length T of the observation sequence. However, an efficient algorithm (linear complexity in T) to evaluate the above expression has been found, based on the principle of optimality (dynamic programming) [Bel03].The algorithm, known as *Forward algorithm* is described in figure 2.15

The decoding problem. The decoding problem consists of finding the most probable sequence of HMM hidden states given a sequence of observations.It is essentially a path-finding optimization problem that will be dealt with using again the dynamic programming paradigm.In fact, the decoding algorithm, also known as the *Viterbi algorithm* (figure 2.16) is very similar to the *Forward* algorithm presented above.

The Viterbi algorithm returns the maximum joint likelihood of the observation and state sequence as well as the corresponding state transition path.The optimal path for a left-to-right HMM, i.e. an HMM where transitions are only allowed in the forward direction, is equivalent to the information required to determine the optimal segmentation of the HMM states to match the observation sequence.

Figure 2.15: *The Forward algorithm for HMM probability evaluation.[JM09]*

function FORWARD(*observations* of len T, *state-graph* of len N) **returns** *forward-prob*

create a probability matrix *forward[N+2,T]*
for each state s **from** 1 **to** N **do** ; initialization step
$forward[s,1] \leftarrow a_{0,s} * b_s(o_1)$
for each time step t **from** 2 **to** T **do** ; recursion step
for each state s **from** 1 **to** N **do**

$$forward[s,t] \leftarrow \sum_{s'=1}^{N} forward[s',t-1] * a_{s',s} * b_s(o_t)$$

$$forward[q_F,\mathrm{T}] \leftarrow \sum_{s=1}^{N} forward[s,T] * a_{s,q_F}$$; termination step

return $forward[q_F,T]$

Figure 2.16: *The Viterbi algorithm for HMM decoding.[YD14]*

function VITERBI(*observations* of len T, *state-graph* of len N) **returns** *best-path*

create a path probability matrix *viterbi[N+2,T]*
for each state s **from** 1 **to** N **do** ; initialization step
$viterbi[s,1] \leftarrow a_{0,s} * b_s(o_1)$
$backpointer[s,1] \leftarrow 0$
for each time step t **from** 2 **to** T **do** ; recursion step
for each state s **from** 1 **to** N **do**

$$viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$$

$$backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s}$$

$$viterbi[q_F,T] \leftarrow \max_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$$; termination step

$$backpointer[q_F,T] \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$$; termination step

return the backtrace path by following backpointers to states back in time from $backpointer[q_F,T]$

Figure 2.17: *The Backward algorithm for HMM decoding.[DHS00]*

1 **initialize** $\omega(T), t = T, a_{ij}, b_{jk}$, visible sequence V^T
2 **for** $t \leftarrow t - 1$;
4 $\beta_j(t) \leftarrow \sum_{i=1}^{c} \beta_i(t+1) a_{ij} b_{jk} v(t+1)$
5 **until** $t = 1$
7 **return** $P(V^T) \leftarrow \beta_i(0)$ for the known initial state
8 **end**

The parameter estimation problem. The goal in HMM training, is to extract the model parameters so as to minimize the empirical risk with respect to the joint likelihood loss, involving a sequence of acoustic data and their corresponding linguistic labels. To estimate the parameters of an HMM model given training data, we will apply the Expectation-Maximization algorithm, also known as *Baum-Welch algorithm* in the context of HMM training.First however, we will introduce the *Backward algorithm*, which is a part of the EM computation for HMMs.

The *Backward algorithm* (figure 2.17) is very similar to the *Forward* but now we are moving backwards, that is, the algorithm computes

$$\beta_t(i) = P(\boldsymbol{o}_{t+1}^T | q_t = i), \ t \ = \ 1, ..., T-1$$

The EM algorithm uses both the Forward and Backward algorithms in the expectation E-step in order to obtain:

- the posterior state transition probabilities in the HMM

$$\xi_t(i,j) = \frac{\alpha_t(i)\beta_{t+1}(j)\alpha_{ij} \exp N_{t+1}(j)}{P(\boldsymbol{o}_1^T|\theta_0)}, \quad t \ = \ 1, ..., T-1$$

  where $N_t(i)$ is the logarithm of the Gaussian p.d.f. associated with state i,

- the posterior state occupancy probabilities

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j)$$

In the maximization M-step the parameters are computed using the current estimates of ξ and γ:

$$\hat{\alpha}_i j = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

$$\hat{\Sigma}_i = \frac{\sum_{t=1}^{T} \gamma_t(i)(\boldsymbol{o}_t - \hat{\boldsymbol{\mu}}_i)(\boldsymbol{o}_t - \hat{\boldsymbol{\mu}}_i)^T}{\sum_{t=1}^{T} \gamma_t(i)}$$

$$\hat{\mu}_i = \frac{\sum_{t=1}^{T} \gamma_t(i)\boldsymbol{o}_t}{\sum_{t=1}^{T} \gamma_t(i)}$$

for each state i.

HMMs in speech modeling and recognition.[YD14][ST04] We have already seen that HMMs can be used as generative models to produce sequences of observations. They are able to produce sequences of variable length, which is of utmost importance for speech modeling and recognition, and they have proven to be good models for the statistical distribution of sequence data of speech acoustics. Consequently, HMMs have become very popular in the ASR community.

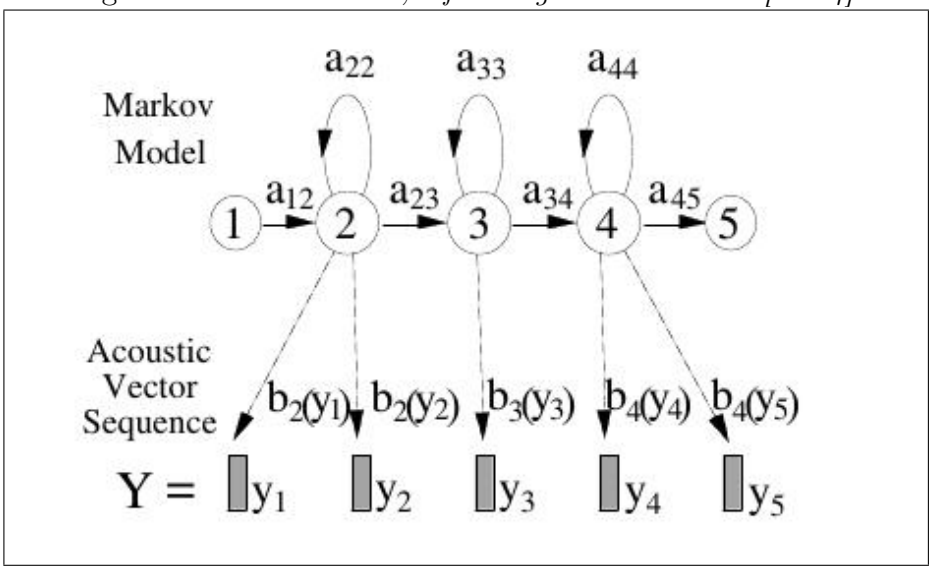
GMM/HMMs in ASR. As we have seen, a GMM/HMM is a statistical model that describes two dependent random processes, an observable and a hidden, where the observable is assumed to be generated by a hidden state according to a Gaussian mixture p.d.f.

In the context of speech, we think of a single HMM as a system generating acoustic features of a modeled speech unit, which can be a word, a syllable, a single phone , or, usually a context-dependent phone. The motivation behind choosing context-dependent phones, and therefore states, stems from the effort to reduce output variability of speech feature vectors associated with each state, leading to more detailed generative modeling. However this leads to an expansion of the state space, which we have seen how it is dealt with, at the beginning of the chapter.

The most common HMM model is the three-state left-to-right HMM (figure 2.18). The number of states is chosen based on the behavior of the vocal tract. It goes through three states when uttering a phone : changing from the previous phone, steady pronunciation of the current phone and changing to the next phone.

However, despite their advantages, HMMs have been found to have several weaknesses. The temporal independence of speech data conditioned on the HMM states and the lack of proven correlation between acoustic features and ways in which speech sounds are produced (e.g. speaking rate and style) have motivated the replacement of GMMs associated with each state by more realistic, temporally correlated dynamic systems containing hidden, continuous-valued dynamic structure (e.g. [Bil03]).

Figure 2.18: *Three-state, left-to-right HMM model.[ST04]*



2.4.5 The Decoder

[MPR01] The components described above are finally combined in the decoder which will find the most likely word sequence given a sequence of feature vectors. In order to define a common framework for the representation and use of the aforementioned models in LVCSR, we represent them by Weighted Finite State Transducers; an approach which provides significant algorithmic and engineering benefits.

A *finite-state transducer* is a finite automaton whose state transitions are labeled with both input and output symbols.Consequently, a path through the transducer encodes a mapping from an input to an output symbol sequence. *Weighted finite-state transducers (WFSTs)* , in addition to input/output symbols, have weights on the transitions which accumulate along paths to compute the total cost of a mapping from an input to an output symbol sequence.Thus, seeing the components of an ASR system as WFSTs and mathematical operations on them, allows for generalizing and efficiently implementing many of the common processing methods in speech recognition. We will briefly present the WFST framework in speech recognition, yet first we will introduce some necessary notation and algorithms.

Notation and Algorithms

- A *semi-ring* $(\mathbb{K}, \bigoplus, \bigodot, \bar{0}, \bar{1})$ is defined by a set of values $\mathbb{K}$, two binary operations of addition ($\bigoplus$) and multiplication ($\bigodot$) and two designated values $\bar{0}$ and $\bar{1}$. The addition operation is associative, commutative and has $\bar{0}$ as the identity element. The multiplication operation is associative, has $\bar{1}$ as the identity element, is distributive with respect to addition and has $\bar{0}$ as the annihilator

Figure 2.19: *Composition algorithm .[MPR01].*

```
WEIGHTED-COMPOSITION(T_1, T_2)
 1  Q ← I_1 × I_2
 2  S ← I_1 × I_2
 3  while S ≠ ∅ do
 4       (q_1, q_2) ← HEAD(S)
 5       DEQUEUE(S)
 6       if (q_1, q_2) ∈ I_1 × I_2 then
 7            I ← I ∪ {(q_1, q_2)}
 8            λ(q_1, q_2) ← λ_1(q_1) ⊗ λ_2(q_2)
 9       if (q_1, q_2) ∈ F_1 × F_2 then
10            F ← F ∪ {(q_1, q_2)}
11            ρ(q_1, q_2) ← ρ_1(q_1) ⊗ ρ_2(q_2)
12       for each (e_1, e_2) ∈ E[q_1] × E[q_2] such that o[e_1] = i[e_2] do
13            if (n[e_1], n[e_2]) ∉ Q then
14                 Q ← Q ∪ {(n[e_1], n[e_2])}
15                 ENQUEUE(S, (n[e_1], n[e_2]))
16            E ← E ∪ {((q_1, q_2), i[e_1], o[e_2], w[e_1] ⊗ w[e_2], (n[e_1], n[e_2]))}
17  return T
```

element: $\forall a \in \mathbb{K}, a \bigodot \bar{0} = \bar{0} \bigodot a = \bar{0}$. If $\bigodot$ is also commutative the semi-ring is called *commutative*, which will be the case for all the semi-rings mentioned later.

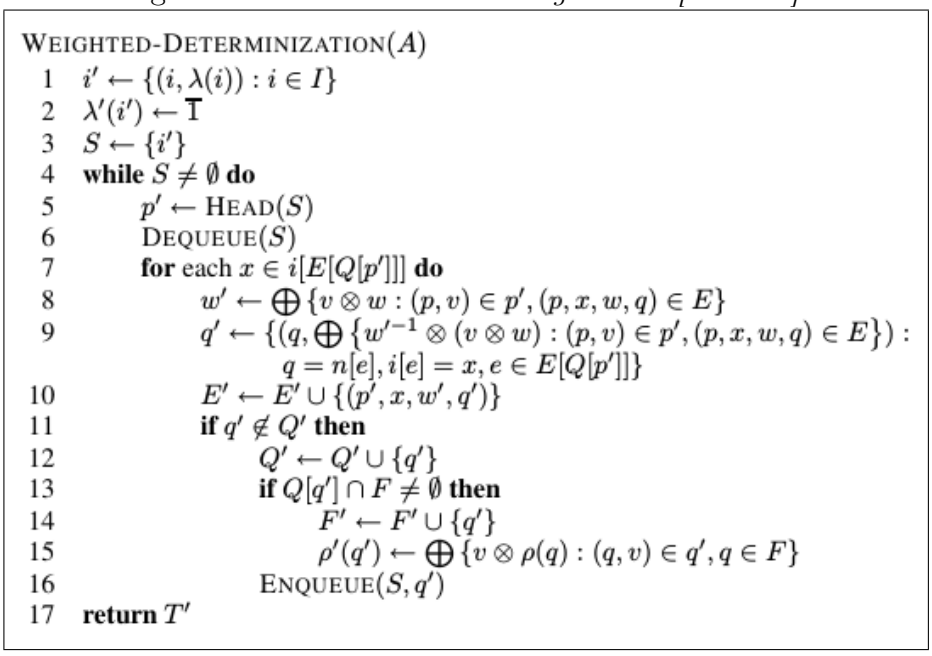
- A WFST $T = (\mathcal{A}, \mathcal{B}, \mathcal{Q}, \mathcal{I}, \mathcal{F}, \mathcal{E}, \lambda, \rho)$ over a semi-ring $\mathbb{K}$ is specified by a finite input alphabet $\mathcal{A}$, a finite output alphabet $\mathcal{B}$, a finite set of states $\mathcal{Q}$, a set of initial states $\mathcal{I} \subseteq \mathcal{Q}$, a set of final states $\mathcal{F} \subseteq \mathcal{Q}$, a finite set of transitions $\mathcal{E} \subseteq Q \times (\mathcal{A} \cup \epsilon) \times (\mathcal{B} \cup \epsilon) \times \mathcal{K} \times \mathcal{Q}$, an initial state weight assignment $\lambda : \mathcal{I} \to \mathcal{K}$ and a final state weight assignment $\rho : \mathcal{F} \to \mathcal{K}$. $\mathcal{E}[q]$ denotes the sum of the number of states and transitions of $\mathcal{T}$.

Based on the notation just introduced we will present the operations on WFSTs that will be used in speech recognition applications.

- *Composition* is the basic operation that allows us to create complex WFSTs from simpler ones, thus putting together all the fundamental components of an ASR system.

- *Determinization* removes non-determinancy from the WFST by ensuring that each state has no more than a single output transition for a given input label.Not every WFST is determinizable, however, there is a *pre-determinization* algorithm that can be used to make determinizable an arbitrary WFST over the tropical semi-ring $((\mathcal{R} \cup \{-\infty, +\infty\}, min, +, +\infty, 0))$ by inserting transitions labeled with special symbols.The determinization operation is particularly important in ASR considering the redundancy found in e.g. the WFST

Figure 2.20: *Determinization algorithm .[MPR01].*



```
WEIGHTED-DETERMINIZATION(A)
 1  i' ← {(i, λ(i)) : i ∈ I}
 2  λ'(i') ← T̄
 3  S ← {i'}
 4  while S ≠ ∅ do
 5      p' ← HEAD(S)
 6      DEQUEUE(S)
 7      for each x ∈ i[E[Q[p']]] do
 8          w' ← ⊕{v ⊗ w : (p, v) ∈ p', (p, x, w, q) ∈ E}
 9          q' ← {(q, ⊕{w'^-1 ⊗ (v ⊗ w) : (p, v) ∈ p', (p, x, w, q) ∈ E}) :
                q = n[e], i[e] = x, e ∈ E[Q[p']]}
10          E' ← E' ∪ {(p', x, w', q')}
11          if q' ∉ Q' then
12              Q' ← Q' ∪ {q'}
13              if Q[q'] ∩ F ≠ ∅ then
14                  F' ← F' ∪ {q'}
15                  ρ'(q') ← ⊕{v ⊗ ρ(q) : (q, v) ∈ q', q ∈ F}
16              ENQUEUE(S, q')
17  return T'
```

representing the pronunciation lexicon.A deterministic lexicon WFST will contain at most one path for any input string, thus less time and space will be needed to process the string (figure 2.20).

- *Minimization* transforms a WFST to an equivalent one with the fewest possible states and transitions which saves both space and time during its processing. Before minimising the transducer, a form of re-weighting (*weight-pushing*) is performed to redistribute weight among transitions as well as to improve search operations.

WFST models in speech recognition

There are four principal models of weighted finite-state transducers that are used in speech recognition:

- G : the word level grammar
- L : the pronunciation lexicon
- C : the context-dependency transducer
- H : the HMM transducer

The word level *grammar transducer G* has a state w_i for each word and transitions are added according to the N-gram model used in the grammar. For example,

a bigram grammar has a transition from state w_1 to w_2 for every bigram w_1w_2 seen in the training corpus.The label of the transition is then w_2 and the weight is the negative logarithm of the transition probability $(-\log(\hat{p}(w_2|w_1)))$.To deal with unseen N-grams while keeping the complexity of constructing the WFST low, we introduce a back-off state b.An unseen bigram w_1w_3 is then represented as two transitions: an ϵ transition w_1b with weight $-\log(\beta(w_1))$ and a transition bw_3 with weight $-\log(\hat{p}(w_3))$. Because ϵ transitions introduce non-determinism in the WFST, we can treat ϵ labels as normal symbols during determinization, thus keeping the number of transitions low - otherwise transitions become quadratic with respect to the size of the vocabulary after determinization.

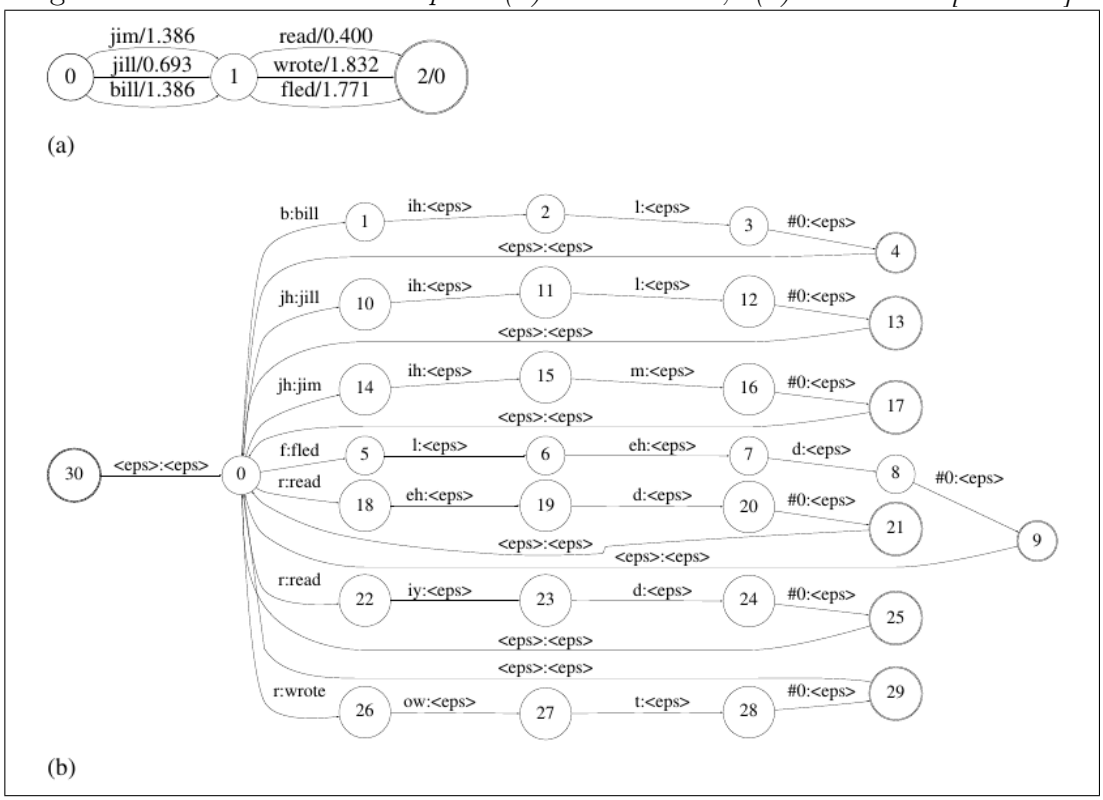
The *pronunciation lexicon transducer* L is the Kleene closure of the union of individual word pronunciations.It is easy to see that L is, in general, not determizable, considering the existence of homophones and the fact that the first word of the output string might be impossible to determine before the entire phone string is scanned. To make L determinizable we add a number of disambiguation symbols $(\#_i)$ as well as a symbol $(\#_0)$ to mark the end of the phonetic transcription of each word. The new lexicon transducer after the addition of these symbols is determinizable and is denoted by $\bar{L}$.

The *context dependency transducer* represents a mapping from context independent phones to context dependent units.The transducer has a state for every pair of phones with label (a, b) ,where a is the past and b is the future phone, and transitions marked as *a:phone/left context_ right context*.To apply the context dependent triphone models often used in ASR in the WFST framework, we need to be able to compose a context dependency transducer with the lexicon transducer introduced earlier.In order to make this composition feasible we first invert the context dependency transducer (interchange input and output labels) and create the transducer C which maps from context dependent triphones to context independent phones.

The final transducer that is used in the decoding process is the *Hidden Markov Models transducer* H which is the closure of the union of the individual HMMs of the acoustic model.

Applying the composition operation on the transducers presented here will output the decoding transducer which we will further optimize to help decoding and make it as efficient as possible.First, composing the lexicon and grammar transducers gives a new transducer that maps from phones to word strings restricted to the grammar $(L \circ G)$.The resulting transducer is then composed with the context dependency transducer C and the resulting transducer maps from context dependent phones to word strings restricted to the grammar $(C \circ L \circ G)$.Finally, $C \circ L \circ G$ is composed with the HMM transducer H and the outcome is a transducer that maps

Figure 2.21: *Transducer example: (a) Grammar G, (b) Lexicon $\bar{L}$ [MPR01].*



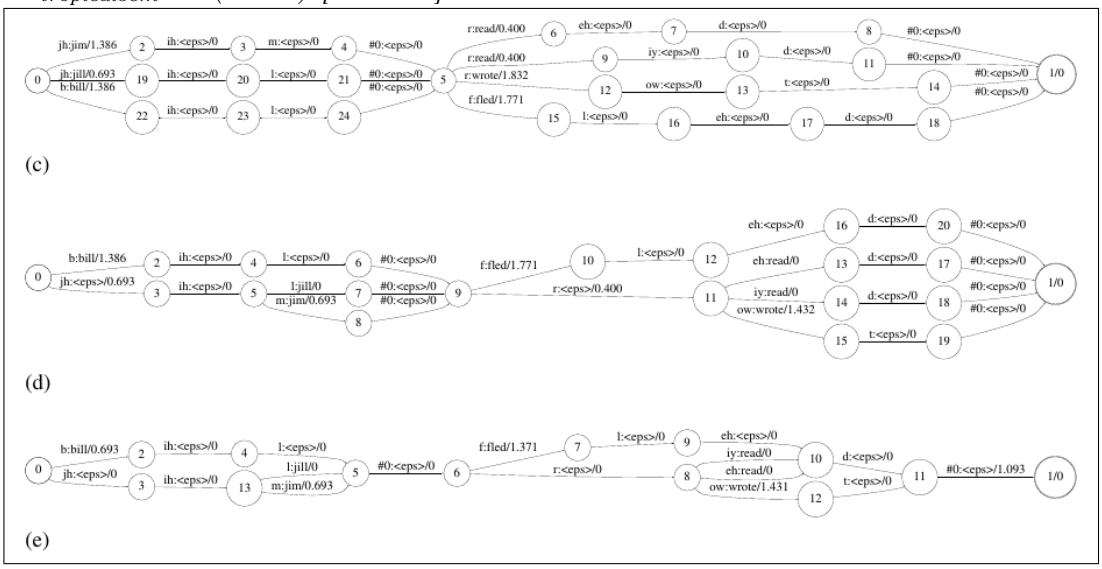
from the identifiers of context-dependent HMM states to word strings restricted to G $(H \circ C \circ L \circ G)$.

After the construction of the final transducer we have to determinize and minimize it so that we have the optimal transducer for recognition.By determinizing it we eliminate redundant paths which reduces recognition time.Moreover, if the determinization is applied after each composition stage during the construction of the graph, the composition operations that follow are performed more efficiently and the total size of the transducer is reduced.Once we have determinized the transducer we can further optimize it by minimization and the weight pushing process that precedes it. As far as weigh pushing is concerned, provided we are using the -log semi-ring, we can have large efficiency gains during the Viterbi beam search, and make sure that all weights leaving a state sum up to one (which is desirable for a language model).

Given the *HCLG* transducer and an utterance of N frames, how do we decode it, namely how do we find the most likely word sequence and its corresponding state-level alignment?

The first step is to construct an acceptor U of the utterance, which is a WFST with identical input and output symbols. The acceptor has $N+1$ states with an arc for each (time,context-dependent HMM state) combination and weights on these

Figure 2.22: *Transducer example: (c)* $\bar{L} \circ G$*, (d)*$\bar{L} \circ G$ *determinized, (e)* $min_{tropicalsem}$ *det(*$\bar{L} \circ G$*) [MPR01].*



arcs the scaled negated acoustic log likelihoods.Following the construction of U, we compose it with the decoding WFST and get a new WFST S:

$$S = U \circ HCLG$$

The decoding problem now reduces to finding the best path through S.The input symbol sequence of the best path is the state-level alignment and the output sequence is the corresponding sentence [PHB+12].

2.4.6 Evaluation

The performance of a large vocabulary continuous speech recognition system is evaluated based on the Word Error Rate (WER) metric. There are three types of possible errors when recognizing continuous speech: (a) substitution errors, i.e. the wrong word is recognized, (b) word deletions, namely the presence of a word is not recognized at all and (c) word insertions, meaning that an extra word is recognized.If we define the number of words in the text speech as N and denote with $C(\cdot)$ the number of errors of each type, then WER is defined as:

$$WER = \frac{C(substitutions) + C(deletions) + C(insertions)}{N}$$

ADD STATE OF THE ART WER + MENTION SYSTEM AND DATASET

Chapter 3

Manifold and Representation learning

In the first chapter we briefly saw an intuitive definition of manifolds. In this chapter we will give a mathematical definition of manifolds and examine the role that they play in representation learning.

Representation learning is the process of learning good representations of the data, that make it easier to extract information that will be useful when we build classifiers or other predictors. A good representation is one that captures the posterior distribution of the latent explanatory factors that led to the observed data and as such, it is particularly effective as input to a supervised predictor [BCV13].

3.1 Manifold learning

[BGC15]

Manifold learning is an approach to representation learning that builds on the *manifold hypothesis*([Cay05]). According to this hypothesis, real-world data presented in high dimensional spaces is expected to concentrate on the vicinity of a manifold $\mathcal{M}$ of much lower dimensionality $d_{\mathcal{M}}$, embedded in high dimensional input space $\mathcal{R}^{d_x}$.

This hypothesis is a kind of prior assumption about the data generating distribution that seems particularly fit to artificial intelligence tasks involving data, such as speech, text, music and images. The common thing about such data, is that if we choose configurations of the observed variables at random, according to a factored distribution (e.g. a uniform distribution), it is very unlikely to generate he kind of observations we want to model. For example, if we uniformly pick values for an acoustic signal, we will most likely create an unnatural speech sample. Consequently, manifold learning algorithms try to discover where probability concentrates in the

input space, as it will be a very small region of the total space of configurations.

Furthermore, making very small changes to the input data, e.g the values of the pixels of an image, will produce natural-looking data similar to the original input. This is another aspect of the manifold hypothesis, that is, probable configurations are likely to be surrounded by other possible configurations.

3.1.1 Mathematical formulation of manifold learning

[Cay05]

We will now give the mathematical definition of manifolds and mathematically formulate manifold learning.

Definition 15. *A homeomorphism is a continuous function whose inverse is also continuous.*

Definition 16. *A d-dimensional manifold $\mathcal{M}$ is a set that is locally homeomorphic with $\mathcal{R}^d$. That is, $\forall\ x \in \mathcal{M}$ there is an open neighborhood around x, $\mathcal{N}_x$, and a homeomophism $f : \mathcal{N}_x \to \mathcal{R}^d$. These neighborhoods are known as coordinate patches and the map as a coordinate chart. The image of the coordinate charts is referred to as the parameter space.*

Intuitively, the dimension d of the manifold indicates the number of independent ways by which a probable configuration can be locally transformed into another probable configuration (remember the example of tiny changes in pixel values of an image). Another concept associated with this intuition is the set of *tangent planes* of a manifold. A tangent plane at a point x on a d-dimensional manifold is given by d basis vectors which cover the local dimensions of variation on the manifold, i.e. they show how x can be changed while staying on the manifold (figure 3.1).

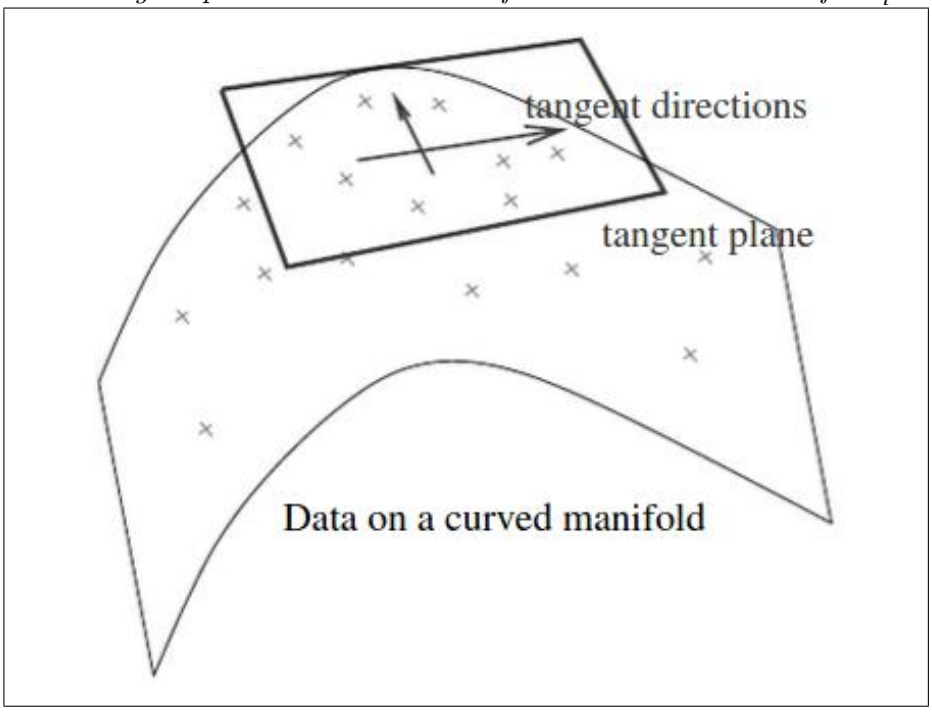
We will only deal with manifolds which are subsets of $\mathcal{R}^D$ where $D >> d$. That is, the manifold will lie in $\mathcal{R}^D$ but will be homeomorphic to a lower dimensional space $\mathcal{R}^d$.

Before moving on to manifold learning we will further constrain the manifolds of interest.

Definition 17. *A smooth or differentiable manifold is a manifold such that each coordinate chart is differentiable with a differentiable inverse. We say that such a chart is a diffeomorphism.*

Definition 18. *An embedding of a manifold $\mathcal{M}$ into $\mathcal{R}^D$ is a smooth homeomoprhism from M to a subset of $\mathcal{R}^D$.*

Figure 3.1: *Tangent plane and directions of variation on the manifold [BGC15].*



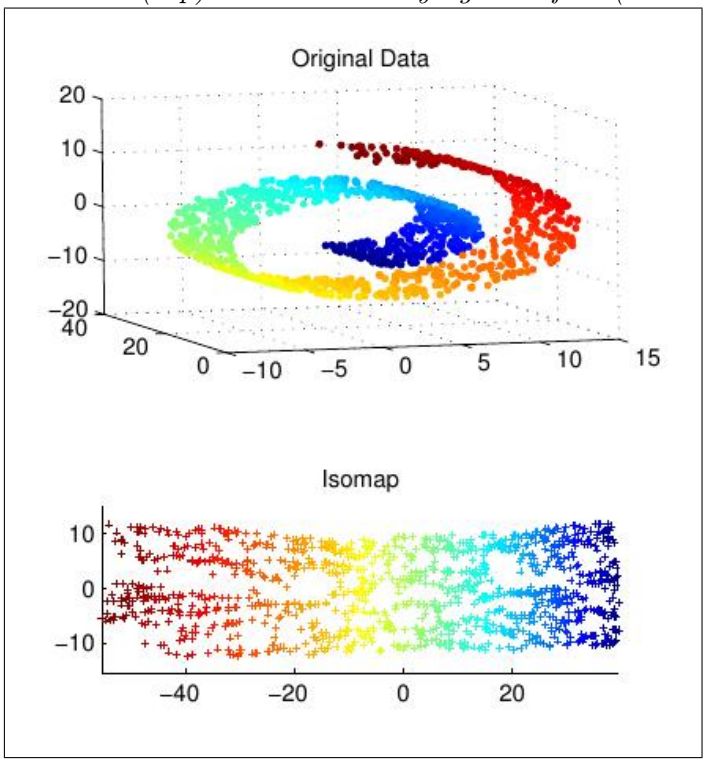
An embedding is an important concept that characterizes a manifold $\mathcal{M}$ and is a representation of the data points on or projected on the manifold. It is usually given by a low-dimensional vector, with less dimensions than the surrounding space, in which the manifold lies. The learning algorithms that will be presented later will learn either directly an embedding for each training data point, or a more general mapping, also called encoder or representation function, that maps any point in the high-dimensional input space to its embedding.

Definition 19. *Suppose we are given a set of $\mathcal{N}$ points $x_1, x_2, ..., x_N \in \mathcal{R}^D$ that lie on a d-dimensional manifold M which can be described by a single coordinate chart $f : \mathcal{M} \to \mathcal{R}^d$. Manifold learning is the process of finding $y_1, y_2, ..., y_N \in \mathcal{R}^d$, where $y_i \doteq f(x_i)$.*

In other words, manifold learning is the process of trying to learn the manifold of a set of available points, or find an embedding of the underlying manifold. The most frequently used example in the manifold literature is the Swiss roll, which is a two-dimensional manifold embedded in the three-dimensional space ($\mathcal{R}^3$). What is important to note is the fact that the distances between points in the original dataset are maintained in the two-dimensional dataset. This is due to the chart f between the two datasets being a homeomorphism.

The fact that the point distances between the datasets are preserved is a basic characteristic of the manifold learning algorithms that will be presented in the next

Figure 3.2: *Swiss roll (top) and the underlying manifold (bottom). [Cay05].*



section.

3.2 Manifold learning algorithms

3.2.1 Dimensionality reduction

[Ver08] We know that most learning algorithms performing well in low-dimensional datasets, perform poorly when applied to high-dimensional datasets. This phenomenon is known as the *curse of dimensionality*. In order to effectively apply learning algorithms in high-dimensional data, we would like first to map it to a low-dimensional space, while preserving much of the important information, and then run the algorithms in the projected space. This process is known as *dimensionality reduction*.

One way to verify the quality of a dimensionality reduction technique is to test how well the mapping preserves pairwise distances. This idea is based on the assumption that the distances between points in space relate to the dissimilarity between the corresponding observations. It is evident that this distance preservation would be much easier if we could find the underlying manifold structure of the data; we would then limit our work on points lying on the manifold and not in the whole ambient space.

A question that arises here, is, whether we can project any dataset on a lower-dimensional space, which maintains structure. The *Johnson-Lindenstrauss lemma* states that any n points in high dimensional euclidean space can be mapped onto k dimensions where $k \geq \mathcal{O}(\frac{\log n}{\epsilon^2})$ without distorting the euclidean distance between any two points more than a factor of $1 \pm \epsilon$ [Mah09], [DG03].

Lemma 2. *For any $0 < \epsilon < 1$ and any integer n, let k be a positive integer such that $k \geq 4(\frac{\epsilon^2}{2} - \frac{\epsilon^3}{3})^{-1} \ln n$.*

Then for any set V of n points in $\mathcal{R}^d$ there is a map $f : \mathcal{R}^d \to \mathcal{R}^k$ such that $\forall u, u \in V$:

$$(1 - \epsilon)\|u - v\|^2 \leq \|f(u) - f(v)\|^2 \leq (1 + \epsilon)\|u - v\|^2$$

This map can be found in randomized polynomial time.

A general outline of the lemma is the following:

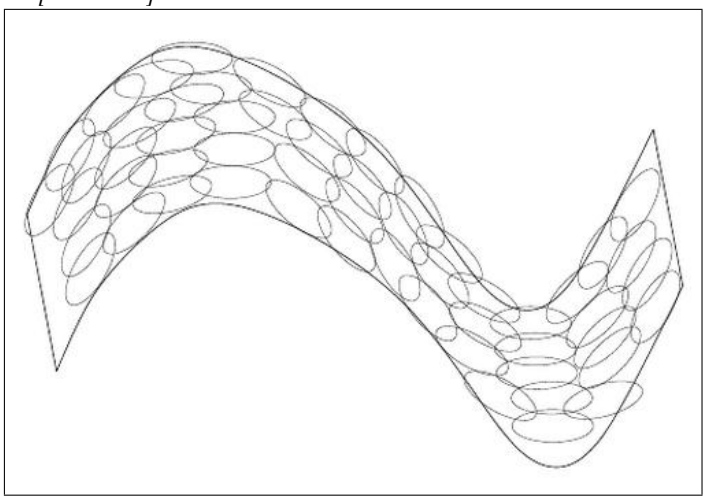
- Construct a random projection over k-dimensional subspaces.
- Prove that the expected value of the euclidean distance of the random projection is equal to the euclidean distance of the orginal subspace.
- Prove that the variance of the euclidean distance is greater than the specified error factor only with a probability $\frac{2}{n^2}$, such that the union bound of this probability across all points is less than $1 - \frac{1}{n}$.

3.2.2 Algorithms

So far, most manifold learning algorithms are unsupervised learning procedures attempting to uncover the underlying manifold structure. They use a nearest-neighbor graph which has one node per training example and edges connecting near neighbors. Each node is associated with a tangent plane which spans the directions of variations associated with a neighborhood of the graph and a coordinate system that associates each training example with an embedding. This coordinate system can be thought of as a local Euclidean system or a locally flat Gaussian, with very small variance in the directions orthogonal to the plane, and a very large variance in the directions defining the local coordinate system (figure 3.3). With this approach, a generalization is possible to new examples by a form of interpolation between neighbors and a global coordinate system can be obtained through an optimization or solving a linear system. Furthermore, by formulating a problem in terms of graph structures one avoids making any assumptions about the data distribution.

However, the methods based on nearest-neighbor graph work well provided we have a very large number of training examples to cover all the curves and twists of

Figure 3.3: *Tangent planes are tiled together to cover the manifold, forming a global coordinate system [BGC15].*



the underlying manifold. This need for huge amounts of data, as well as the fact that manifolds of interest in AI, such as speech and images, have many curves and twists, has motivated the use of deep learning methods and distributed representations to uncover the manifold structure. These methods try to learn a coordinate system for the main latent factors that explain the structure of the underlying generating distribution.

We will now introduce some basic manifold learning algorithms.

Isomap

Isometric feature mapping (*ISOMAP*) was one of the first manifold learning algorithms. Isomap contains the following three steps:

- Construct the k nearest-neighbor graph. The number of neighbors k is defined after trying with various values and comparing the results. It is also possible to find the neighbors with an approximate nearest-neighbor procedure, e.g. use neighborhoods with a certain ϵ radius.

- Estimate the distances along the manifold (geodesic distances) between input points using shortest-path distances on the neighbor graph constructed in the previous step.

  Isomap assumes that there is an isometric chart mapping the input points to the low-dimensional space, i.e. it preserves the distances between points. If the manifold is smooth enough then the geodesic distance between nearby points is almost linear. Given that the manifold locally resembles the Euclidean space, the Euclidean distance can be used to estimate distances. On the other hand,

Figure 3.4: *Classical Multidimensional Scaling [Cay05].*

classical Multidimensional Scaling
input: $D \in \mathbb{R}^{n \times n}(D_{ii} = 0,\ \ D_{ij} \geq 0),\ \ d \in \{1, \ldots, n\}$

1. Set $B := -\frac{1}{2}HDH$, where $H = I - \frac{1}{n}\mathbf{1}\mathbf{1}^\top$ is the centering matrix.
2. Compute the spectral decomposition of B: $B = U\Lambda U^\top$.
3. Form Λ_+ by setting $[\Lambda_+]_{ij} := \max\{\Lambda_{ij}, 0\}$.
4. Set $X := U\Lambda_+^{1/2}$.
5. Return $[X]_{n \times d}$.

for points that are far apart from each other, we cannot estimate distances in the same way, because the manifold structure is not the same globally as it is locally. To compute such distances we use shortest-paths algorithms on the neighbors graph we have constructed.

- Using Multidimensional Scaling find points in lower-dimensional Euclidean space whose interpoint distances match the geodesic distances found in the previous step.

  In the third step Isomap finds points whose Euclidean distances equal the geodesic distances found earlier. Supposing that the manifold is isometrically embedded, we are certain that such points exist and are unique to translation and rotation. To find such points we apply a technique called *Multidimensional scaling* (figure 3.4), which, given a matrix $D \in \mathcal{R}^{n \times v}$ of dissimilarities constructs a set of points whose interpoint Euclidean distances match closely those in D.

Furthermore, Isomap has the following two important properties. First, it automatically estimates the dimensionality of the underlying manifold: the number of non-zero eigenvalues found by MDS is the underlying dimensionality.

Second, under the following assumptions, Isomap is guaranteed to recover the parameterization of a manifold.

- The manifold is isometrically embedded into $\mathcal{R}^D$.
- The underlying parameter space is convex, or intuitively, the parameter space of the manifold cannot contain any holes.
- The manifold is compact and well-sampled everywhere.

Figure 3.5: *ISOMAP [Cay05].*

Isomap
input: $x_1, \ldots, x_n \in \mathbb{R}^D$, k

1. Form the k-nearest neighbor graph with edge weights $W_{ij} := \|x_i - x_j\|$ for neighboring points x_i, x_j.
2. Compute the shortest path distances between all pairs of points using Dijkstra's or Floyd's algorithm. Store the squares of these distances in D.
3. Return $Y :=$cMDS(D).

Locally Linear Embedding

The intuition behind Locally Linear Embedding (LLE) stems from thinking of the manifold as a collection of overlapping cooordinate patches. Then, is the neighborhood sizes are small and the manifold does not have many curves or twists, then these patches will be approximately linear. The concept is to find these patches, discover their geometry and find a mapping from the manifold to $\mathcal{R}^d$ that preserves the local geometry and is approximately linear. Since these patches are overlapping, the local reconstructions will combine into a global one.

The first step of LLE tries to model the manifold as a collection of linear patches and uncover their geometry. In order to do so, it represents each point x_i as a weighted, convex combination of its nearest neighbors. The weights are chose based on minimizing the squared error:

$$\|x_i - \sum_{j \in N(x_i)} W_{ij} x_j\|^2$$

where $N(i)$ is the set of nearest-neighbors of x_i.

The weight matrix W reveals the layout of the neighbors around each point; thus it is W that captures the local geometry of each patch. Apart from the convexity, we impose the following constraint on the weights:

$$W_{ij} = 0 \; if \; j \notin N(i)$$

These two constraints have an important physical meaning: the convexity makes the weights invariant to global rotations, translations and scalings, e.g.

$$\|x_i + a - \sum_{j \in N(x_i)} W_{ij}(x_j + a)\|^2 = \|x_i - \sum_{j \in N(x_i)} W_{ij} x_j\|^2$$

whereas the second enforces locality on the patches.

The above minimization problem has a closed-form solution for the weight matrix which, for each x_i is given by:

$$\hat{W}_i = \frac{\sum_k C_{jk}^{-1}}{\sum_{lm} C_{lm}^{-1}}$$

where C is the local covariance matrix

$$C_{jk} \doteq (x_i - \eta_j)^T (x_i - \eta_k)$$

and η_j, η_k are neighbors of x_i. $\hat{W} \in \mathcal{R}^{n \times k}$ is then transformed into the sparse $W \in \mathcal{R}^{n \times n}$ by setting $W_{ij} = \hat{W}_{il}$ if x_j is the l^{th} neighbor of x_i and $W_{ij} = 0$ if $j \notin N(i)$.

The second step in LLE is to find a configuration of points in the d-dimensional parameter space whose local geometry is described well by W. Contrary to Isomap LLE cannot discover the number of dimensions d; we have to provide it based on prior knowledge about the parameter space or estimate it using other techniques (e.g. Conformal Eigenmaps, [SS05]). Such a configuration is found by minimizing the cost function:

$$\sum_i \|y_i - \sum_j W_{ij} y_j\|^2 = Y^T [(I - W)^T (I - W)] Y = Y^T M Y$$

with respect to $y_1, y_2, ..., y_n \in \mathcal{R}^d$ and under constraints: $Y^T Y = I$ and $\sum_i Y_i = \mathbf{0}$. The first constraint forces the solution to be of rank d whereas the second centers the embedding around the beginning of the axes. Minimizing the cost function corresponds to setting the columns of Y to the eigenvectors that correspond to the minimum d eigenvalues of $\mathcal{M}$. However, the minimum (eigenvector,eigenvalue) pair is ($\mathbf{1}$,0). Therefore, to avoid the last coordinate being identical for all points, we pick the minimum d eigenvectors that are not constant.

Laplacian Eigenmaps

The Laplacian Eigenmaps algorithm is based on the concept of the Laplacian of a graph: Given a graph and the corresponding matrix containing edge weights, the Laplacian of the graph is defined as $L \doteq D - W$, where D is defined is the diagonal matrix defined as $D_{ii} = \sum_j W_{ij}$. The eigenvalues of L reveal information about the structure of the graph, such as whether it is complete or connected. Since in our case the weight matrix contains information about the neighborhoods of the points, its Laplacian will help us discover local information about the underlying manifold.

Figure 3.6: *LLE [Cay05].*

Locally Linear Embedding
input: $x_1, \ldots, x_n \in \mathbb{R}^D$, d, k

1. *Compute reconstruction weights.* For each point x_i, set
$$W_i := \frac{\sum_k C_{jk}^{-1}}{\sum_{lm} C_{lm}^{-1}}.$$
2. *Compute the low-dimensional embedding.*
   - Let U be the matrix whose columns are the eigenvectors of $(I - W)^\top (I - W)$ with nonzero accompanying eigenvalues.
   - Return $Y := [U]_{n \times d}$.

In Laplacian Eigenmaps we have to choices for constructing for the weight matrix W.Either use the simple scheme:

$$W_{ij} = \begin{cases} 1 & \text{if } j \in N(i) \\ 0 & \text{if } j \notin N(i) \end{cases}$$

or use the Gaussian heat kernel:

$$W_{ij} = \begin{cases} e^{\frac{-\|x_i - x_j\|^2}{2\sigma^2}} & \text{if } j \in N(i) \\ 0 & \text{if } j \notin N(i) \end{cases}$$

Like LLE we know use W to find points in the d-dimensional parameter space (d is set or estimated in advance) that preserve the relations of their original counterparts.

The cost function we are aiming to minimize now is based on the fact that we want to maintain the distance between points:

$$\sum_{ij} W_{ij} \|y_i - y_j\|^2 = tr(Y^T L Y)$$

To avoid getting to a solution with less than d dimensions, we enforce the constraint $Y^T D Y = I$. In this way we will ensure that the rank of Y is d. We now have to solve the generalized eigenvalue problem $L\boldsymbol{y} = \lambda D\boldsymbol{y}$ and populate Y with the eigenvectors that correspond to the d smallest, non-zero eigenvalues.

3.3 Discriminative manifold learning algorithms in ASR

The reason why we presented some basic ideas and algorithms coming from the area of manifold learning is the fact that speech sounds are suspected to lie on

Figure 3.7: *Laplacian Eigenmaps [Cay05].*

Laplacian Eigenmaps
input: $x_1 \dots x_n \in \mathbb{R}^D$, d, k, σ.

1. Set $W_{ij} = \begin{cases} e^{-\|x_i - x_j\|^2/2\sigma^2} & \text{if } x_j \in N(i) \\ 0 & \text{otherwise.} \end{cases}$
2. Let U be the matrix whose columns are the eigenvectors of $L\mathbf{y} = \lambda D\mathbf{y}$ with nonzero accompanying eigenvalues.
3. Return $Y := [U]_{n \times d}$.

a low-dimensional manifold, which we can exploit to reduce the dimensionality of speech feature vectors. Approaches to automatic speech recognition using manifold learning is an active field of research that has so far produced promising results.

3.3.1 Manifolds and speech data

We had mentioned earlier that natural data is usually high dimensional, yet the latent factors that generate it may lie in lower-dimensional spaces. Intuitively one can see why the manifold assumption is true for certain classes of speech: given that there is only a small finite number of articulatory systems involved (tracheia, glottis, vocal tract, tongue, lips, teeth), it is reasonable to expect the existence of a low-dimensional structure in speech data.

In their work, Jansen and Niyogi ([JN05]) consider wave propagation in acoustic tube models and present a derivation of the sounds generated by it. They represent steady-state sounds produced by such models with the magnitude of their Fourier coefficients and thus, as a point in a high dimensional space. They prove that the set of all sounds generated, as the configuration of the articulatory system varies, lies on a low dimensional manifold embedded in the ambient high-dimensional space.

We will outline the analysis in their paper ([JN05]) when a single tube is modeling the vocal tract.

In order to be able to arrive at a tractable problem they introduce the following approximations: the vocal tract walls are rigid, their impedance is much greater than that of air and the transverse dimension of the vocal tract is much smaller than the signal's wavelength. Based on these assumptions the problem of acoustically analysing the vocal tract resonator comes down to the analysis of planar waves in one spatial dimension.

The notation used is:

- p is the air pressure
- ρ is the density
- u is the velocity
- $A(x)$ is the cross-section area of the tube as a function of the position
- $U = Au$ is the volume velocity of the air perturbations

Starting from the continuity equation for compressible fluid flow and the preservation of momentum, we can arrive at a partial differential equation with respect to the volume velocity U as a function of space and time, which can be further reduced to the standard wave equation in free space:

$$\frac{\partial^2 U}{\partial x^2} = \frac{1}{c^2}\frac{\partial^2 U}{\partial t^2}$$

or, using pressure p

$$\frac{\partial^2 p}{\partial x^2} = \frac{1}{c^2}\frac{\partial^2 p}{\partial t^2}$$

with the boundary conditions

$$U(0,t) = s(t)$$

where $s(t)$ represents the glottal vibrations as a function of time, and

$$\hat{U}(L,\omega) = \frac{\hat{p}(L,\omega)}{\boldsymbol{Z}_r(\omega)}$$

where the second condition is in the frequency domain and $\boldsymbol{Z}_r(\omega)$ is the acoustic impedance at the open end of the tube ($x = L$).

The solution $U(x,t)$ to the above equation can be expressed using its Fourier transform $\hat{U}(x,\omega)$:

$$U(x,t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \hat{U}(x,\omega)e^{i\omega t}d\omega$$

Substituting in the wave equation and solving we get the general solution for the volume and (after working in a similar way) pressure respectively:

$$\hat{U}(x,\omega) = U_1(\omega)e^{-ikx} + U_2(\omega)e^{ikx}$$

$$\hat{p}(x,\omega) = p_1(\omega)e^{-ikx} + p_2(\omega)e^{ikx}$$

The detailed computations for the magnitudes can be found in [JN05]. Eventually we arrive at the general solutions for U and p:

$$\hat{U}(x,\omega) = \hat{s}(\omega)\left[\frac{e^{-ikx}}{1+\frac{B+1}{B-1}e^{-i2kL}} + \frac{e^{ikx}}{1+\frac{B-1}{B+1}e^{i2kL}}\right]$$

$$\hat{p}(x,\omega) = \hat{s}(\omega)\boldsymbol{Z}_r(\omega)\left[\frac{e^{-ikx}}{1+\frac{B+1}{B-1}e^{-i2kL}} + \frac{e^{ikx}}{1+\frac{B-1}{B+1}e^{i2kL}}\right]$$

where $B = \frac{\rho_0 c}{A\boldsymbol{Z}_r(\omega)}$, ρ_0 is the equilibrium density air value.

Since any odd periodic function can be expressed as a Fourier series of sinusoidal sources, we pick a single-frequency sinusoidal as the source function:

$$s(t) = U_0 sin\omega_0 t$$

If we compute its Fourier transform, substitute in the previous solutions and take the inverse Fourier, we arrive at the final form of the solution in the time-domain:

$$p(L,t) = \frac{U_0}{2i}\Big[f(\omega_0,L,A)e^{i\omega_0 t} - f(-\omega_0,L,A)e^{-i\omega_0 t} = Im(U_0 f(\omega_0,L,A)e^{i\omega_0 t})$$

where $f(\omega,L,A) = \boldsymbol{Z}_r(\omega)\Big[coskL - i\frac{A\boldsymbol{Z}_r(\omega)}{\rho_0 c}sinkL\Big]^{-1}$ and we arrived at the second form given that $\boldsymbol{Z}_r(\omega) = \boldsymbol{Z}_r^*(-\omega)$ and consequently $f^*(\omega,L,A) = f(-\omega,L,A)$.

If we use a linear combination of H harmonics as the source function

$$s(t) = \sum_{n=1}^{H} a_n sin(n\omega_0 t)$$

then at the output we expect a solution of the form

$$p(L,t) = \sum_{n=1}^{H} \beta_n sin(n\omega_0 t + \phi_n)$$

where for each n

$$\beta_n sin(n\omega_0 t + \phi_n) = Im(\alpha_n f(n\omega_0,L,A)e^{in\omega_0 t})$$

Consequently, the output Fourier coefficients β_n are given by

$$\beta_n(L,A) = \alpha_n |f(n\omega_0,L,A)|$$

If we now define a subset of $\mathcal{R}^H$ for a given set $\alpha_i | i = 1,...,H]$ by

$$\mathcal{M}_1(L_1,L_2) = (\beta_1,\beta_2,...,\beta_H) | L \in (L_1,L_2)$$

then $\mathcal{M}$ traces-out a one-dimensional curve in the ambient Fourier space $\mathcal{R}^H$ with the following properties:

- There exists a diffeomorphism $\phi : (L_1,L_2) \in \mathcal{R}^1 \rightarrow \mathcal{M}_1(L_1,L_2) \in \mathcal{R}^H$ for L_1, L_2 in the range of human vocal tract lengths.

- The diffeomorphism $\phi^{-1} : \mathcal{M}_1(L_1, L_2) \to \mathcal{R}^1$ is a coordinate chart on the set $\mathcal{M}_1(L_1, L_2)$.
- The set $\mathcal{M}_1(L_1, L_2)$ is open.

These properties tell us that the set $\mathcal{M}_1(L_1, L_2)$ is a smooth and open one dimensional manifold embedded in $\mathcal{R}^H$, which is not a straight line and spans the whole ambient space.

The manifold structure of data can be exploited in several ways: improve supervised learning techniques by applying regularization based on the manifold of speech data (*manifold regularization*) and find new methods of dimensionality reduction. One such method will be presented next.

3.3.2 Discriminative manifold learning

[TR14][TR13a]

In the context of automatic speech recognition we usually want to classify our data into the corresponding phoneme categories, e.g. biphones or triphones. It has recently been established ([SR03]) that the geometric and local structure of the space the data lie on, is important for classification tasks. Consequently, if we could identify the underlying manifold of the available data and apply a discriminative feature transformation, we should see an improvement in our classification efforts.

However, discriminative transformations alone are incapable of capturing the geometric and local distributional structure of the data space, whereas on the other hand, manifold learning algorithms that do uncover the geometric properties of the space, are unsupervised and non-discriminative. Rose and Tomar propose a new framework ([TR14]) that introduces a discriminative element in manifold learning techniques: they aim to maximize the separability between different classes while at the same time preserving the manifold-constraint relationships between data points belonging to the same class.

Their discriminative manifold learning framework starts -similarly to the already presented techniques- by embedding feature vectors into two high-dimensional graphs connecting neighborhoods of vectors, followed by an optimization of their structure according to certain constraints. These weighted, undirected graphs characterize the class-specific manifolds the feature vectors lie on. One of them, called the *intrinsic* graph, defines the relationships between same-class feature vectors ($\mathcal{G}_{int} = \{\boldsymbol{X}, \boldsymbol{W}_{int}\}$), whereas the other, called *penalty* graph, defines the relationships between feature vectors belonging to different classes ($\mathcal{G}_{pen} = \{\boldsymbol{X}, \boldsymbol{W}_{pen}\}$).

The characteristics of the two graphs, such as structure, connectivity and compactness, are mostly influenced by the weights on their edges, which are calculated

according to the distance between points. Therefore, intuitively we understand that the graphs express the geometry of the data space.

Based on the metrics used to calculate distance, the writers propose two different approaches, both resulting in the estimation of a projection matrix $\mathcal{P} \in R^{d \times m}, m \ll d$, which maximizes the sub-manifold class discrimination in the projected space while at the same time it preserves the within-sub-manifold intrinsic data relations.

Locality Preserving Discriminant Analysis

[TR12a]

In Locality Preserving Discriminant Analysis (*LPDA*) the distance metric used is the Euclidean distance. The elements of the weight matrices $\boldsymbol{W}_{int}$ and $\boldsymbol{W}_{pen}$ are calculated using the Gaussian heat kernel:

$$w_{ij}^{int} = \begin{cases} e^{\frac{-\|x_i - x_j\|^2}{\rho}} & \text{if } C(x_i) = C(x_j),\ e(x_i, x_j) = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$w_{ij}^{pen} = \begin{cases} e^{\frac{-\|x_i - x_j\|^2}{\rho}} & \text{if } C(x_i) \neq C(x_j),\ e(x_i, x_j) = 1 \\ 0 & \text{otherwise} \end{cases}$$

where ρ is the heat kernel scale parameter, $C(x_i)$ refers to the class of vector x_i and $e(x_i, x_j) = 1$ indicates that x_i is in the near neighborhood of x_j.The neighbors of vector x_i can be discovered either by an exact or by an approximate k-nearest neighbors method, such as a range search for points lying in a ball of radius r around x_i.

In essence, the meaning of the aforementioned weights assignment is that in graph $\mathcal{G}_{int}$ a node x_i is connected to its k_{int}-nearest neighbors having the same label, whereas in $\mathcal{G}_{pen}$ a node x_i is connected to its k_{pen}-nearest neighbors that have a different label than itself. The values of k_{int} and k_{pen} are determined empirically.

We can define a scatter measure for a graph G as follows:

$$F_G(\boldsymbol{P}) = \sum_{i \neq j} \|y_i - y_j\|^2 w_{ij} = 2\boldsymbol{P^T X(D - W)X^T P}$$

where $\boldsymbol{D}$ is a diagonal matrix defined as $D_{ii} = \sum_j w_{ij}$. Depending on whether we want to retain or discard graph properties, the optimal projection matrix $\boldsymbol{P}$ can be obtained by minimizing or maximizing F_G.

Specifically for LPDA, one would want to reinforce the fact that points belonging to different classes are conceptually far away from each other and on the other hand strengthen the relations between points of the same class. Intuitively it follows that

one should aim to maximize the scatter of the penalty graph $F_{pen}(\boldsymbol{P})$ and minimize the scatter of the intrinsic graph $F_{int}(\boldsymbol{P})$. To combine the two scatters into a single measure the writers define their ratio as a measure of class-separability and graph preservation:

$$F(\boldsymbol{P}) = \frac{F_p(\boldsymbol{P})}{F_i(\boldsymbol{P})} = \frac{\boldsymbol{P}^T\boldsymbol{X}(\boldsymbol{D}_p - \boldsymbol{W}_p)\boldsymbol{X}^T\boldsymbol{P}}{\boldsymbol{P}^T\boldsymbol{X}(\boldsymbol{D}_i - \boldsymbol{W}_i)\boldsymbol{X}^T\boldsymbol{P}}$$

where i denotes *intrinsic* and p *penalty.*

The optimal $\boldsymbol{P}$ is the one to maximize $F(\boldsymbol{P})$:

$$\boldsymbol{P}_{LPDA} = \underset{\boldsymbol{P}}{\operatorname{argmax}} F(\boldsymbol{P}) = \underset{\boldsymbol{P}}{\operatorname{argmax}} \left\{ tr\Big((\boldsymbol{X}(\boldsymbol{D}_i - \boldsymbol{W}_i)\boldsymbol{X}^T\boldsymbol{P})^{-1}(\boldsymbol{P}^T\boldsymbol{X}(\boldsymbol{D}_p - \boldsymbol{W}_p)\boldsymbol{X}^T\boldsymbol{P}) \Big) \right\}$$

This maximization problem can be brought down to solving the following generalized eigenvalue problem:

$$(\boldsymbol{X}(\boldsymbol{D}_p - \boldsymbol{W}_p)\boldsymbol{X}^T)\boldsymbol{p}^j_{lpda} = \lambda_j(\boldsymbol{X}(\boldsymbol{D}_i - \boldsymbol{W}_i)\boldsymbol{X}^T)\boldsymbol{p}^j_{lpda}$$

where $\boldsymbol{p}^j_{lpda}$ is the j^{th} column of the transformation matrix $\boldsymbol{P}_{lpda} \in R^{d\times m}$ and is the eigenvector associated with the j_{th} largest eigenvalue. Therefore $\boldsymbol{P}^{optimal}_{lpda}$ contains the m eigenvectors associated with the m largest eigenvalues.

Correlation Preserving Discriminant Analysis

[TR13b][TR12b]

The second approach, Correlation Preserving Discriminant Analysis (CPDA), uses the cosine-correlation, i.e. normalized inner-product, as distance metric. This is motivated by the fact that models based on Euclidean distance are susceptible to noise, whereas cosine-correlation based models are expected to be less influenced by noise, given the robustness of the angles between cepstrum vectors.

Similarly to LPDA, CPDA uses two undirected graphs , ($\mathcal{G}_{int} = \{\boldsymbol{X}, \boldsymbol{W}_{int}\}$), ($\mathcal{G}_{pen} = \{\boldsymbol{X}, \boldsymbol{W}_{pen}\}$), to embed the feature vectors, yet this time the similarity between nodes is calculated based on the cosine-correlation distance:

$$w^{int}_{ij} = \begin{cases} e^{\frac{\langle x_i, x_j\rangle - 1}{\rho}} & \text{if } C(x_i) = C(x_j),\ e(x_i, x_j) = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$w^{pen}_{ij} = \begin{cases} e^{\frac{\langle x_i, x_j\rangle - 1}{\rho}} & \text{if } C(x_i) \neq C(x_j),\ e(x_i, x_j) = 1 \\ 0 & \text{otherwise} \end{cases}$$

where the notation is the same as above. The scatter measure used in CPDA is the same as in LDPA, yet CPDA first projects the data points onto the surface of a unit hypersphere, thus discarding magnitude information and keeping correlation-based information about data points. Thus, the projection matrix $\mathcal{P} \in R^{d\times m}, m \ll$

d, eventually estimated, will non-linearly project the features from the original d-dimensional hypersphere onto the target m-dimensional hypersphere.

The scatter for CPDA is:

$$F_G(\boldsymbol{P}) = \sum_{i \neq j} \|y_i - y_j\|^2 w_{ij} \overset{\boldsymbol{y}=\boldsymbol{P}^T\boldsymbol{x}/\|\boldsymbol{P}^T\boldsymbol{x}\|}{=} \sum_{i \neq j} \left\| \frac{\boldsymbol{P}^T\boldsymbol{x}_i}{\|\boldsymbol{P}^T\boldsymbol{x}_i\|} - \frac{\boldsymbol{P}^T\boldsymbol{x}_j}{\|\boldsymbol{P}^T\boldsymbol{x}_j\|} \right\| w_{ij} = 2 \sum_{i \neq j} \left(1 - \frac{f_{ij}}{f_i f_j}\right) w_{ij}$$

where for two arbitrary vectors x_u, x_v:

$$f_u = \sqrt{x_u^T \boldsymbol{P}\boldsymbol{P}^T x_u}$$

and

$$f_{uv} = \sqrt{x_u^T \boldsymbol{P}\boldsymbol{P}^T x_v}$$

As previously, we want to maximize the scatter of the penalty graph and minimize the scatter of the intrinsic graph. However, in CPDA the writers have chosen the difference of the two scatter measures as a measure of manifold separability and graph preservation:

$$F(\boldsymbol{P}) = F_{pen}(\boldsymbol{P}) - F_{int}(\boldsymbol{P}) = 2 \sum_{i \neq j} (1 - \frac{f_{ij}}{f_i f_j}) w_{ij}^{p-i}$$

where $w_{ij}^{p-i} = w_{ij}^{pen} - w_{ij}^{int}$. The optimal projection matrix $\boldsymbol{P}_{cpda}^{opt}$ is obtained by maximizing $F(\boldsymbol{P})$:

$$\boldsymbol{P}_{CPDA} = \underset{\boldsymbol{P}}{\operatorname{argmax}}\, F(\boldsymbol{P})$$

Because of the projection of the feature vectors onto the unit hypersphere, the maximization problem cannot be solved by solving an equivalent generalized eigenvalue problem. Therefore, an iterative procedure is chosen, the well known gradient ascent:

$$P := P + \alpha \nabla_P F$$

where

$$\nabla_P F = 2 \sum_{i \neq j} \Big[\frac{f_{ij} x_i x_i^T}{f_i^3 f_j} + \frac{f_{ij} x_j x_j^T}{f_i f_j^3} - \frac{x_i x_j^T + x_j x_i^T}{f_i f_j} \Big] \boldsymbol{P} w_{ij}^{p-i}$$

where α is the gradient scaling factor and $\nabla_P F$ represents the gradient of the scatter measure F with respect to $\boldsymbol{P}$. Given that F is non-linear and non-convex, a good initialization is important to avoid reaching just a local maximum. Such an initialization can be obtained by neglecting the projection on the unit hypersphere, approximate a linear transformation:

$$\boldsymbol{y}_i = \boldsymbol{P}^T x_i$$

and eventually acquire $\boldsymbol{P}_{init}$ by solving the generalized eigenvalue problem:

$$(\boldsymbol{X}(\boldsymbol{D}_p - \boldsymbol{W}_p)\boldsymbol{X}^T)\boldsymbol{p}^j = \lambda_j(\boldsymbol{X}(\boldsymbol{D}_i - \boldsymbol{W}_i)\boldsymbol{X}^T)\boldsymbol{p}^j$$

where $\boldsymbol{D}$ is a diagonal matrix with elements $D_{ii} = \sum_i w_{ij}$ and $\boldsymbol{X}$ contains the normalized feature vectors. As before, $\boldsymbol{P}_{cpda}^{init}$ is composed of the eigenvectors corresponding to the m largest eigenvalues.

Noise Aware Manifold Learning

[TR13c]

Furthermore, in their work Rose and Tomar explore a new method to increase noise robustness of manifold learning methods. They point out that since edge weights, which depend on the Gaussian heat kernel size and shape, characterize local neighborhoods, there is a strong relation between kernel size and the robustness to background noise.

The kernel size governs the compactness of the neighborhood and the smoothness of the manifolds, and in turn, it is governed by the scale parameter ρ. Therefore, the choice of ρ is important to the definition of local neighborhoods and thus to the characteristics of the transformation. If ρ had a value that is too large, then the kernel would tend to flatten, and all data pairs on the graph would have the same importance; on the other hand, if it had a too small value, the manifold would lack smoothing and thus the kernel would be too sensitive to noise. Therefore, the writers claim that the optimal value for ρ is dependent on the SNR level of the speech signal, and support their claims with successful experimental results.

They propose that multiple scale parameters be used, each specific to a noise condition. First, during training, they gather multiple SNR dependent LPDA or CPDA projections and Continuous Density HMMs. This procedure is broken down into three steps:

- Based on a set of ρ values, determine the set of the corresponding projection matrices
- Heuristically identify the optimal value of rho and the corresponding transformation that maximizes ASR performance for a given SNR level
- Train CDHMM models from the features obtained using the set of projection matrices previously acquired

Having this set of SNR-dependent LPDA/CPDA transformations, SNR is estimated for each test utterance and the feature space projection is performed using the corresponding LPDA/CPDA projection matrix. Eventually, the corresponding CDHMM model is used in the ASR application.

This procedure is referred to as Noise-Aware LPDA/CPDA (*N-LPDA* or *N-CPDA*).

Results

All of the proposed techniques are verified by experiments the writers conducted using the Aurora-2 and Aurora-4 datasets.

LPDA and CPDA show a relative WER improvement ranging from 6 to 30% compared to conventional techniques, such as Linear Discriminant Analysis or Locality Preserving Projections. This verifies the assertion that manifold and discriminative learning result in a well-behaved and robust feature-space transformation.

Furthermore, CPDA performs better that LPDA under high noise conditions, supporting the noise robustness of cosine-correlation distance compared to Euclidean. As far as Noise-Aware manifold learning is concerned, N-LPDA performs slightly better on average as compared to LPDA for most noise conditions (0.10-0.20 reduced WER).

At this point it is important to point out the huge computational complexity involved in computing the weight matrices, given the high dimensionality of the feature space. To tackle this issue the writers propose using novel approximate nearest-neighbors techniques such as Locality Sensitive Hashing ([TR13b]).

Bibliography

[BCV13] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8):1798–1828, August 2013.

[BDVJ03] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, March 2003.

[Bel03] Richard Ernest Bellman. *Dynamic Programming*. Dover Publications, Incorporated, 2003.

[BGC15] Yoshua Bengio, Ian J. Goodfellow, and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2015.

[Bil03] Jeff A. Bilmes. Buried markov models: a graphical-modeling approach to automatic speech recognition. *Computer Speech and Language*, 17(2.3):213 – 231, 2003. New Computational Paradigms for Acoustic Modeling in Speech Recognition.

[Cay05] L. Cayton. Algorithms for manifold learning. *Univ. of California at San Diego Tech. Rep*, 2005.

[DG03] Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of johnson and lindenstrauss. *Random Struct. Algorithms*, 22(1):60–65, January 2003.

[DHS00] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2Nd Edition)*. Wiley-Interscience, 2000.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 39(1):1–38, 1977.

[JM09] Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2Nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2009.

[JN05] A. Jansen and P. Niyogi. A geometric perspective on speech sounds. Technical report, 2005.

[Kre78] E. Kreyszig. *Introductory Functional Analysis With Applications*. Wiley Classics Library. John Wiley & Sons, 1978.

[Mah09] Michael Mahoney. Algorithms for massive datasets analysis, lecture1 notes, 2009.

[MKB+10] Tomas Mikolov, Martin Karafiat, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur. Recurrent neural network based language model. In Takao Kobayashi, Keikichi Hirose, and Satoshi Nakamura, editors, *INTERSPEECH*, pages 1045–1048. ISCA, 2010.

[MPR01] Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted finite-state transducers in speech recognition, 2001.

[PHB+12] Daniel Povey, Mirko Hannemann, Gilles Boulianne, Lukas Burget, Arnab Ghoshal, Milos Janda, Martin Karafiat, Stefan Kombrink, Petr Motlicek, Yanmin Qian, Korbinian Riedhammer, Karel Vesely, and Ngoc Thang Vu. Generating exact lattices in the wfst framework. In *ICASSP*, pages 4213–4216. IEEE, 2012.

[Rab89] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *PROCEEDINGS OF THE IEEE*, pages 257–286, 1989.

[RJ86] L. Rabiner and B. Juang. An introduction to hidden markov models. *IEEE Acoutics, Speech and Signal Processing Magazine*, 3:4–16, 1986.

[SR03] Lawrence K. Saul and Sam T. Roweis. Think globally, fit locally: Unsupervised learning of low dimensional manifolds. *J. Mach. Learn. Res.*, 4:119–155, December 2003.

[SS05] Fei Sha and Lawrence K. Saul. Analysis and extension of spectral methods for nonlinear dimensionality reduction. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 784–791, New York, NY, USA, 2005. ACM.

[ST04] Darius Silingas and Laimutis Telksnys. Specifics of hidden markov model modifications for large vocabulary continuous speech recognition. *Informatica, Lith. Acad. Sci.*, 15(1):93–110, 2004.

[SVN37] S. S. Stevens, J. Volkmann, and E. B. Newman. A scale for the measurement of the psychological magnitude pitch. *The Journal of the Acoustical Society of America*, 8(3):185–190, 1937.

[TR12a] Vikrant Singh Tomar and Richard C. Rose. Application of a locality preserving discriminant analysis approach to ASR. In *11th International Conference on Information Science, Signal Processing and their Applications, ISSPA 2012, Montreal, QC, Canada, July 2-5, 2012*, pages 103–107, 2012.

[TR12b] Vikrant Singh Tomar and Richard C. Rose. A correlational discriminant approach to feature extraction for robust speech recognition. In *INTERSPEECH 2012, 13th Annual Conference of the International Speech Communication Association, Portland, Oregon, USA, September 9-13, 2012*, pages 555–558, 2012.

[TR13a] Vikrant Singh Tomar and Richard C. Rose. Efficient manifold learning for speech recognition using locality sensitive hashing. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*, pages 6995–6999, 2013.

[TR13b] Vikrant Singh Tomar and Richard C. Rose. Locality sensitive hashing for fast computation of correlational manifold learning based feature space transformations. In *INTERSPEECH 2013, 14th Annual Conference of the International Speech Communication Association, Lyon, France, August 25-29, 2013*, pages 1776–1780, 2013.

[TR13c] Vikrant Singh Tomar and Richard C. Rose. Noise aware manifold learning for robust speech recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*, pages 7087–7091, 2013.

[TR14] Vikrant Singh Tomar and Richard C. Rose. A family of discriminative manifold learning algorithms and their application to speech recognition. *IEEE/ACM Transactions on Audio, Speech & Language Processing*, 22(1):161–171, 2014.

[Ver08] Nakul Verma. Mathematical advances in manifold learning, 2008.

[YD14] Dong Yu and Li Deng. *Automatic Speech Recognition: A Deep Learning Approach*. Springer Publishing Company, Incorporated, 2014.

[YEK+02] S. Young, G. Evermann, D. Kershaw, G. Moore, J. Odell, D. Ollason, V. Valtchev, and P. Woodland. The htk book. *Cambridge University Engineering Department*, 3, 2002.

[You02] Steve Young. Acoustic modelling for large vocabulary continuous speech recognition. *Cambridge University Engineering Department*, 2002.