There was a celebrated Fourier at the Academy of Science, whom posterity has forgotten; and in some garret an obscure Fourier, whom the future will recall.

Victor Hugo Les Misérables

Jean Baptiste Joseph Fourier (1768-1830) studied the mathematical theory of heat conduction in his major work, The Analytic Theory of Heat, (Théorie analytique de la chaleur). He established the partial differential equation governing heat diffusion and solved it using an infinite series of trigonometric functions. The description of a signal in terms of elementary trigonometric functions had a profound effect on the way signals are analysed. The Fourier method is the most extensively applied signal-processing tool. This is because the transform output lends itself to easy interpretation and manipulation, and leads to the concept of frequency analysis. Furthermore even biological systems such as the human auditory system perform some form of frequency analysis of the input signals. This chapter begins with an introduction to the complex Fourier series and the Fourier transform, and then considers the discrete Fourier transform, the Fast Fourier transform, the 2-D Fourier transform and the discrete cosine transform. Important engineering issues such as the trade-off between the time and frequency resolutions, problems with finite data length, windowing and spectral leakage are considered. The applications of the Fourier transform include filtering, telecommunication, music processing, pitch modification, signal coding and signal synthesis feature extraction for pattern identification as in speech recognition, image processing, spectral analysis in astrophysics, radar signal processing.
3.1 Introduction

The objective of signal transformation is to express a signal as a combination of a set of basic “building block” signals, known as the basis functions. The transform output should lend itself to convenient analysis, interpretation and manipulation. A useful consequence of transforms, such as the Fourier and the Laplace, is that differential analysis on the time domain signal become simple algebraic operations on the transformed signal. In the Fourier transform the basic building block signals are sinusoidal signals with different periods giving rise to the concept of frequency. In Fourier analysis a signal is decomposed into its constituent sinusoids, i.e. frequencies, the amplitudes of various frequencies form the so-called frequency spectrum of the signal. In an inverse Fourier transform operation the signal can be synthesised by adding up its constituent frequencies. It turns out that many signals that we encounter in daily life such as speech, car engine noise, bird songs, music etc. have a periodic or quasi-periodic structure, and that the cochlea in the human hearing system performs a kind of harmonic analysis of the input audio signals. Therefore the concept of frequency is not a purely mathematical abstraction in that biological and physical systems have also evolved to make use of the frequency analysis concept.

The power of the Fourier transform in signal analysis and pattern recognition is its ability to reveals spectral structures that may be used to characterise a signal. This is illustrated in Fig. 3.1 for the two extreme cases of a sine wave and a purely random signal. For a periodic signal the power is concentrated in extremely narrow bands of frequencies indicating the existence of structure and the predictable character of the signal. In the case of a pure sine wave as shown in Fig. 3.1.a the signal power is concentrated in one frequency. For a purely random signal as shown in Fig 3.1.b the signal power is spread equally in the frequency domain indicating the lack of structure in the signal.

![Figure 3.1](image-url)

**Figure 3.1** The concentration or spread of power in frequency indicates the correlated or random character of a signal: (a) a predictable signal, (b) a random signal.
Notation: In this chapter the symbols $t$ and $m$ denote continuous and discrete
time variables, and $f$ and $k$ denote continuous and discrete frequency variables
respectively. The variable $\omega=2\pi f$ denotes the angular frequency in units of rad/s
and is used interchangeably (within a scaling of factor of $2\pi$) with the frequency
variable $f$ in units of Hz.

3.2 Fourier Series: Representation of Periodic Signals

The following three sinusoidal functions form the basis functions for the Fourier
analysis
\[
x_1(t) = \cos \omega_0 t \tag{3.1}
\]
\[
x_2(t) = \sin \omega_0 t \tag{3.2}
\]
\[
x_3(t) = \cos \omega_0 t + j \sin \omega_0 t = e^{j \omega_0 t} \tag{3.3}
\]

Fig. 3.2.a shows the cosine and the sine components of the complex exponential
(cisoidal) signal of Eq. (3.3), and Fig. 3.2.b shows a vector representation of the
complex exponential in a complex plane with real (Re) and imaginary (Im)
dimensions. The Fourier basis functions are periodic with an angular frequency of
$\omega_0$ rad/s and a period of $T_0=2\pi/\omega_0=1/F_0$ seconds, where $F_0$ is the frequency in
Hz. The following properties make the sinusoids the ideal choice as the
elementary building block basis functions for signal analysis and synthesis:

(i) Orthogonality; two sinusoidal functions of different frequencies have the
following orthogonal property:

Figure 3.2 - Fourier basis functions: (a) real and imaginary parts of a complex sinusoid,
(b) vector representation of a complex exponential.
\[
\int_{-\infty}^{\infty} \sin(\omega_1 t) \sin(\omega_2 t) \, dt = -\frac{1}{2} \int_{-\infty}^{\infty} \cos(\omega_1 + \omega_2) t \, dt + \frac{1}{2} \int_{-\infty}^{\infty} \cos(\omega_1 - \omega_2) t \, dt = 0
\]

(3.4)

For harmonically related sinusoids the integration can be taken over one period. Similar equations can be derived for the product of cosines, or sine and cosine, of different frequencies. Orthogonality implies that the sinusoidal basis functions are independent and can be processed independently. For example in a graphic equaliser we can change the relative amplitudes of one set of frequencies, such as the bass, without affecting other frequencies, and in subband coding different frequency bands are coded independently and allocated different number of bits.

(ii) Sinusoidal functions are infinitely differentiable. This is important, as most signal analysis, synthesis and manipulation methods require the signals to be differentiable.

(iii) Sine and cosine signals of the same frequency have only a phase difference of \(\pi/2\) or equivalently a relative time delay of a quarter of one period i.e. \(T_0/4\).

Associated with the complex exponential function \(e^{j\omega t}\) is a set of harmonically related complex exponential of the form

\[
[1, e^{\pm j\omega_0}, e^{\pm 2j\omega_0}, e^{\pm 3j\omega_0}, \ldots]
\]

(3.5)

The set of exponential signals in Eq. (3.5) are periodic with a fundamental frequency \(\omega_0=2\pi/T_0=2\pi F_0\) where \(T_0\) is the period and \(F_0\) is the fundamental frequency. These signals form the set of basis functions for the Fourier analysis. Any linear combination of these signals of the form

\[
\sum_{k=\infty}^{\infty} c_k e^{j\omega_0 t}
\]

(3.6)

is also periodic with a period of \(T_0\). Conversely any periodic signal \(x(t)\) can be synthesised from a linear combination of harmonically related exponentials. The Fourier series representation of a periodic signal are given by the following synthesis and analysis equations:

\[
x(t) = \sum_{k=-\infty}^{\infty} c_k e^{j\omega_0 t}, \quad k = \cdots -1,0,1,\ldots \quad \text{Synthesis equation (3.7)}
\]
Sec. 3.2 Fourier Series

\[ c_k = \frac{1}{T_0} \int_{-T_0/2}^{T_0/2} x(t) e^{-j2\pi k t / T_0} dt \quad k = \cdots -1, 0, 1, \cdots \]  

Analysis equation (3.8)

The complex-valued coefficient \( c_k \) conveys the amplitude (a measure of the strength) and the phase of the frequency content of the signal at \( k \omega_0 \) Hz. Note from the analysis Eq. (3.8), that the coefficient \( c_k \) may be interpreted as a measure of the correlation of the signal \( x(t) \) and the complex exponential \( e^{-j \omega_0 t} \).

The set of complex coefficients \( \cdots c_{-1}, c_0, c_1, \cdots \) are known as the signal spectrum. Eq. (3.7) is referred to as the synthesis equation, and can be used as a frequency synthesizer (as in music synthesizers) to generate a signal as a weighted combination of its elementary frequencies. The representation of a signal in the form of Eq. (3.7) as the sum of its constituent harmonics is also referred to as the complex Fourier series representation. Note from Eqs. (3.7) and (3.8) that the complex exponentials that form a periodic signal occur only at discrete frequencies which are integer multiples, i.e. harmonics, of the fundamental frequency \( \omega_0 \).

Therefore the spectrum of a periodic signal, with a period of \( T_0 \), is discrete in frequency with discrete spectral lines spaced at integer multiples of \( \omega_0 = 2\pi / T_0 \).

**Example 3.1** Given the Fourier synthesis Eq. (3.7), obtain the frequency analysis Eq. (3.8).

**Solution:** Multiply both sides of Eq. (3.7) by \( e^{-j \omega_0 t} \) and integrate over one period to obtain

\[ \int_{-T_0/2}^{T_0/2} x(t) e^{-j2\pi k t / T_0} dt = \sum_{k=-\infty}^{\infty} c_k \int_{-T_0/2}^{T_0/2} e^{j2\pi k t / T_0} e^{-j2\pi m t / T_0} dt = \sum_{k=-\infty}^{\infty} c_k \int_{-T_0/2}^{T_0/2} e^{j2\pi (k-m) t / T_0} dt \]  

(3.9)

From the orthogonality principle the integral in the r.h.s of Eq. (3.9) is zero unless \( k=m \) in which case the integral is equal to \( T_0 \). Hence

\[ c_m = \frac{1}{T_0} \int_{-T_0/2}^{T_0/2} x(t) e^{-j2\pi m t / T_0} dt \]  

(3.10)
Example 3.2 Find the frequency spectrum of a 1 kHz sinewave shown in Fig 3.3.a

\( x(t) = \sin(2000\pi t) \quad -\infty < t < \infty \) \hfill (3.11)

Solution A: The Fourier synthesis Eq. (3.7) can be written as

\[
x(t) = \sum_{k=-\infty}^{\infty} c_k e^{j2000\pi t} = \cdots + c_{-1} e^{-j2000\pi t} + c_0 + c_1 e^{j2000\pi t} + \cdots \quad (3.12)
\]

now the sine wave can be expressed as

\[
x(t) = \sin(2000\pi t) = \frac{1}{2j} e^{j2000\pi t} - \frac{1}{2j} e^{-j2000\pi t} \quad (3.13)
\]

Equating the coefficients of Eqs. (3.12) and (3.13) yields

\[
c_1 = \frac{1}{2j}, \quad c_{-1} = -\frac{1}{2j} \quad \text{and} \quad c_{k \neq \pm 1} = 0 \quad (3.14)
\]

Fig 3.3.b shows the magnitude spectrum of the sinewave, where the spectral lines \( c_1 \) and \( c_{-1} \) correspond to the 1 kHz and \(-1\) kHz frequencies respectively.

Solution B: Substituting \( \sin(2000\pi t) = \frac{1}{2j} e^{j2000\pi t} - \frac{1}{2j} e^{-j2000\pi t} \) in the Fourier analysis Eq. (3.8) yields

\[
c_k = \frac{1}{T_0} \int_{-T_0/2}^{T_0/2} \left( \frac{1}{2j} e^{j2000\pi t} - \frac{1}{2j} e^{-j2000\pi t} \right) e^{-jk2000\pi t} dt
\]

\[
= \frac{1}{2jT_0} \int_{-T_0/2}^{T_0/2} e^{j(1-k)2000\pi t} dt - \frac{1}{2jT_0} \int_{-T_0/2}^{T_0/2} e^{-j(1+k)2000\pi t} dt \quad (3.15)
\]
Since sine and cosine functions are positive-valued over one half a period and negative-valued over the other half, it follows that Eq. (3.15) is zero unless \( k=1 \) or \( k=-1 \).

\[
c_1 = \frac{1}{2j} \quad \text{and} \quad c_{-1} = -\frac{1}{2j} \quad \text{and} \quad c_{k \neq \pm 1} = 0
\]

(3.16)

**Example 3.3** Find the frequency spectrum of a periodic train of pulses with amplitude of 1.0, a period of 1.0 kHz and a pulse 'on' duration of 0.3 milliseconds.

**Solution:** The pulse period \( T_0=1/F_0=0.001 \) s, and the angular frequency \( \omega_0=2\pi F_0=2000\pi \) rad/s. Substituting the pulse signal in the Fourier analysis Eq. (3.8) gives

\[
c_k = \frac{1}{T_0} \int_{-T_0/2}^{T_0/2} x(t) e^{-j\omega_0 t} dt = \frac{1}{0.001} \int_{-0.00015}^{0.00015} e^{-j\cdot 2000\pi t} dt
\]

(3.17)

For \( k=0 \) as \( c_0=\sin(0)/0 \) is undefined, differentiate the numerator and denominator of Eq. (3.17) w.r.t. to the variable \( k \) (strictly this can only be done for a continuous variable \( k \) i.e. when the period \( T_0 \) tends to infinity) to obtain

\[
c_0 = \frac{0.3\pi \cos(0.3\pi 0)}{\pi} = 0.3
\]

(3.18)

![Figure 3.4 - A rectangular pulse train and its discrete frequency 'line' spectrum.](image)
Example 3.4 For the example 3.3 write the formula for synthesising the signal up to the $N^{th}$ harmonic, and plot a few examples for the increasing number of harmonics.

Solution: The equation for the synthesis of a signal up to the $N^{th}$ harmonic content is given by

$$x(t) = \sum_{k=-N}^{N} c_k e^{jk\omega_0 t} = c_0 + \sum_{k=1}^{N} c_k e^{jk\omega_0 t} + \sum_{k=-N}^{-1} c_k e^{-jk\omega_0 t}$$

$$= c_0 + \sum_{k=1}^{N} [\text{Re}(c_k) + j\text{Im}(c_k)](\cos(k\omega_0 t) + j\sin(k\omega_0 t))$$

$$+ \sum_{k=1}^{N} [\text{Re}(c_k) - j\text{Im}(c_k)](\cos(k\omega_0 t) - j\sin(k\omega_0 t))$$

$$= c_0 + \sum_{k=1}^{N} [2\text{Re}(c_k)\cos(k\omega_0 t) - 2\text{Im}(c_k)\sin(k\omega_0 t)]$$

(3.19)

The following MatLab code generates a synthesised pulse train composed of $N$ harmonics. In this example there are 5 cycles in an array of 1000 samples. Fig. 3.5 shows the waveform for the number of harmonics equal to; 1, 3, 6, and 100.

```
NHarmonics=100; Ncycles=5; Nsamples=1000;
y(1:Nsamples)=0.3; j=1:Nsamples;
for k=1:NHarmonics
    x(j)=(2*sin(0.3*pi*k)/(pi*k))* cos(k*2*pi*Ncycles*j/Nsamples);
y=y+x;
end
```

Figure 3.5 Illustration of the Fourier synthesis of a periodic pulse train, and the Gibbs phenomenon, with the increasing number of harmonics in the Fourier synthesis: (a) $N=1$, (b) $N=3$, (c) $N=6$, and (d) $N=100$. 
3.2.1 Fourier Synthesis of Discontinuous Signals: Gibbs Phenomenon

The sinusoidal basis functions of the Fourier transform are smooth and infinitely differentiable. In the vicinity of a discontinuity the Fourier synthesis of a signal exhibits ripples as shown in the Fig 3.5. The peak amplitude of the ripples does not decrease as the number of harmonics used in the signal synthesis increases. This behaviour is known as the Gibbs phenomenon. For a discontinuity of unity height, the partial sum of the harmonics exhibits a maximum value of 1.09 (that is an overshoot of 9%) irrespective of the number of harmonics used in the Fourier series. As the number of harmonics used in the signal synthesis increases, the ripples become compressed toward the discontinuity but the peak amplitude of the ripples remains constant.

3.3 Fourier Transform: Representation of Aperiodic Signals

The Fourier series representation of periodic signals consist of harmonically related spectral lines spaced at the integer multiples of the fundamental frequency. The Fourier representation of aperiodic signals can be developed by regarding an aperiodic signal as a special case of a periodic signal with an infinite period. If the period of a signal is infinite, then the signal does not repeat itself and is aperiodic. Now consider the discrete spectra of a periodic signal with a period of $T_0$ as

![Figure 3.6](image)

Figure 3.6 – (a) A periodic pulse train and its line spectrum. (b) a single pulse from the periodic train in (a) with an imagined ‘off’ duration of infinity; its spectrum is the envelope of the spectrum of the periodic signal in (a).
shown in Fig. 3.6.a. As the period $T_0$ is increased, the fundamental frequency $F_0 = 1/T_0$ decreases, and successive spectral lines become more closely spaced. In the limit as the period tends to infinity (i.e., as the signal becomes aperiodic) the discrete spectral lines merge and form a continuous spectrum. Therefore, the Fourier equations for an aperiodic signal, (known as the Fourier transform), must reflect the fact that the frequency spectrum of an aperiodic signal is continuous. Hence to obtain the Fourier transform relation the discrete-frequency variables and operations in the Fourier series Eqs. (3.7) and (3.8) should be replaced by their continuous-frequency counterparts. That is the discrete summation sign $\Sigma$ should be replaced by the continuous summation integral $\int$, the discrete harmonics of the fundamental frequency $kF_0$ should be replaced by the continuous frequency variable $f$, and the discrete frequency spectrum $c_k$ must be replaced by a continuous frequency spectrum say $\mathcal{X}(f)$. The Fourier synthesis and analysis equations for aperiodic signals, the so-called Fourier transform pair, are given by

$$x(t) = \int_{-\infty}^{\infty} X(f)e^{j2\pi ft} \, df$$

(3.20)

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi ft} \, dt$$

(3.21)

Note from Eq. (3.21), that $X(f)$ may be interpreted as a measure of the correlation of the signal $x(t)$ and the complex sinusoid $e^{-j2\pi ft}$. The condition for existence and computability of the Fourier transform integral of a signal $x(t)$ is that the signal must have finite energy

$$\int_{-\infty}^{\infty} |x(t)|^2 \, dt < \infty$$

(3.22)

**Example 3.5** Derivation of inverse Fourier transform

Given the Fourier transform Eq. (3.21) derive the Fourier synthesis Eq. (3.20).

**Solution:** Consider the Fourier analysis Eq. (3.8) for a periodic signal and Eq. (3.21) for its non-periodic version (consisting of one period only). Comparing these equations reproduced below

$$c_k = \frac{1}{T_0} \int_{-T_0/2}^{T_0/2} x(t)e^{-j2\pi kft} \, dt$$

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi ft} \, dt$$

we have
where $F_0 = 1/T_0$. Using Eq. (3.23) the Fourier synthesis Eq. (3.7) for a periodic signal can be rewritten as

$$x(t) = \sum_{k=-\infty}^{\infty} X(k) e^{j2\pi kF_0 t} \Delta F$$  \hspace{1cm} (3.24)$$

where $\Delta F = 1/T_0 = F_0$ is the frequency spacing between successive spectral lines of the spectrum of a periodic signal as shown in Fig. 3.6. Now as the period $T_0$ tends to infinity, $\Delta F = 1/T_0$ tends to zero, then the discrete frequency variables $kF_0$ and $\Delta F$ should be replaced by a continuous frequency variable $f$, and the discrete summation sign by the continuous integral sign. Thus Eq. (3.24) becomes

$$x(t) \xrightarrow{T_0 \to \infty} \int_{-\infty}^{\infty} X(f) e^{j2\pi ft} df$$  \hspace{1cm} (3.25)$$

### Example 3.6 The spectrum of an Impulse Function

Consider the unit-area pulse $p(t)$ shown in Fig 3.7.a. As the pulse width $\Delta$ tends to zero the pulse tends to an impulse. The impulse function shown in Fig 3.7.b is defined as a pulse with an infinitesimal time width as

$$\delta(t) = \lim_{\Delta \to 0} p(t) = \begin{cases} 1/\Delta & |t| \leq \Delta/2 \\ 0 & |t| > \Delta/2 \end{cases}$$  \hspace{1cm} (3.26)$$
It is easy to see that the integral of the impulse function is given by

$$\int_{-\infty}^{\infty} \delta(t) dt = \Delta \times \frac{1}{\Delta} = 1$$

(3.27)

The important **sampling property of the impulse function** is defined as

$$\int_{-\infty}^{\infty} x(t) \delta(t - T) dt = x(T)$$

(3.28)

The Fourier transform of the impulse function is obtained as

$$\Delta(f) = \int_{-\infty}^{\infty} \delta(t) e^{-j2\pi ft} dt = \delta^0 = 1$$

(3.29)

The impulse function is used as a **test function** to obtain the impulse response of a system. This is because as shown in Fig 3.7.c an impulse is a spectrally rich signal containing all frequencies in equal amounts.

**Example 3.7** Find the spectrum of the 10 kHz periodic pulse train shown in Fig. 3.8.a with an amplitude of 1 mV and expressed as

$$x(t) = \sum_{m=\infty}^{\infty} A \times \text{limit}_{\Delta \to 0} p(t - mT_0) \times \Delta = \sum_{m=\infty}^{\infty} A \delta(t - mT_0)$$

(3.30)

where $p(t)$ is a unit-area pulse with width $\Delta$ as shown in Fig. 3.7.a, and the function $\delta(t-mT_0)$ is now assumed to have a unit amplitude representing the area under the impulse.

**Solution:** $T_0 = 1/F_0 = 0.1$ ms and $A = 1$ mV. For the time interval $-T_0/2 < t < T_0/2$ $x(t) = A \delta(t)$, hence we have

**Figure 3.8** A periodic impulse train $x(t)$ and its Fourier series spectrum $c_k$. 
Sec. 3.3 Fourier Transform

\[ c_k = \frac{1}{T_0} \int_{-T_0/2}^{T_0/2} x(t) e^{-j2\pi f_k t} dt = \frac{1}{T_0} \int_{-T_0/2}^{T_0/2} A \delta(t) e^{-j2\pi f_k t} dt = A \frac{T_0}{T_0} = 10 \text{ V} \quad (3.31) \]

As shown in Fig. 3.8.b the spectrum of a periodic impulse train in time is a periodic impulse train in frequency.

**Example 3.8 The Spectrum of a Rectangular Function: Sinc Function**

The rectangular pulse is particularly important in digital signal analysis and digital communication. For example, the spectrum of a rectangular pulse can be used to calculate the bandwidth required by pulse radar systems or communication systems that transmit pulses. The Fourier transform of a rectangular pulse Fig. 3.9.a of duration \( T \) seconds is obtained as

\[ R(f) = \int_{-\infty}^{\infty} r(t) e^{-j2\pi ft} dt = \int_{-T/2}^{T/2} e^{-j2\pi ft} dt \]
\[ = \frac{e^{j2\pi fT/2} - e^{-j2\pi fT/2}}{j2\pi f} = T \frac{\sin(\pi fT)}{\pi f} = T \text{sinc}(\pi fT) \quad (3.32) \]

Fig. 3.9.b shows the spectrum of the rectangular pulse. Note that most of the pulse energy is concentrated in the main lobe within a bandwidth of \( 2/T \). However there are pulse energy in the side lobes that may interfere with other electronic devices operating at the side lobe frequencies.

Matlab code for drawing the spectrum of a rectangular pulse.

```matlab
% Pulsewidth T, Number of frequency samples N, Frequency resolution for plot df.
T=.001;df=10;N=1000;
for i=1:N
    if (i~=N/2)x(i)=T*sin(2*pi*df*T*(i-N/2))/(2*pi*df*T*(i-N/2));end
end
x(N/2)=T; plot(x);
```

---

**Figure 3.9** A rectangular pulse and its spectrum.
3.3.1 The Relation Between the Laplace and the Fourier Transforms

The Laplace transform of $x(t)$ is given by the integral

$$X(s) = \int_{0^{-}}^{\infty} x(t)e^{-st} dt$$

(3.33)

where the complex variable $s = \sigma + j\omega$, and the lower limit of $t = 0^-$ allows the possibility that the signal $x(t)$ may include an impulse. The inverse Laplace transform is defined by

$$x(t) = \int_{\sigma_1 + j\omega}^{\sigma_1 - j\omega} X(s)e^{st} ds$$

(3.34)

where $\sigma_1$ is selected so that $X(s)$ is analytic (no singularities) for $s > \sigma$. The basis functions for the Laplace transform are damped or growing sinusoids of the form $e^{-\sigma t} = e^{-\sigma} e^{-j\omega t}$ as shown in Fig. 3.10. These are particularly suitable for transient signal analysis. The Fourier basis functions are steady complex exponential, $e^{-j\omega t}$, of time-invariant amplitudes and phase, suitable for steady state or time-invariant signal analysis.

The Laplace transform is a one-sided transform with the lower limit of integration at $t = 0^-$, whereas the Fourier transform Eq. (3.21) is a two-sided transform with the lower limit of integration at $t = -\infty$. However for a one-sided signal, which is zero-valued for $t < 0^-$, the limits of integration for the Laplace and the Fourier transforms are identical. In that case if the variable $s$ in the Laplace transform is replaced with the frequency variable $j\omega$ then the Laplace integral becomes the Fourier integral. Hence for a one-sided signal, the Fourier transform is a special case of the Laplace transform corresponding to $s = j\omega$ and $\sigma = 0$. The relation between the Fourier and the Laplace transforms are discussed further in Chapter 4.

3.3.2 Properties of the Fourier Transform
There are a number of Fourier Transform properties that provide further insight into the transform and are useful in reducing the complexity of the solutions of Fourier transforms and inverse transforms. These are:

**Linearity**
The Fourier transform is a linear operation, this means the principle of superposition applies. Hence if:

\[ z(t) = ax(t) + by(t) \]  

then

\[ Z(f) = aX(f) + bY(f) \]  

**Symmetry**
This property states that if the time domain signal \( x(t) \) is real (as is often the case in practice) then

\[ X(f) = X^*(-f) \]  

Where the superscript asterisk \( ^* \) denotes the complex conjugate operation. From Eq. (3.37) it follows that \( \text{Re}\{X(f)\} \) is an even function of \( f \) and \( \text{Im}\{X(f)\} \) is an odd function of \( f \). Similarly the magnitude of \( X(f) \) is an even function and the phase angle is an odd function.

**Time Shifting and Frequency Modulation (FM)**
Let \( X(f) = \mathcal{F}[x(t)] \) be the Fourier transform of \( x(t) \). If the time domain signal \( x(t) \) is delayed by an amount \( T_0 \), the effect on its spectrum \( X(f) \) is a phase shift of \( e^{-j2\pi f T_0} \) as

\[ \mathcal{F}[x(t - T_0)] = e^{-j2\pi f T_0} X(f) \]  

Conversely if \( X(f) \) is shifted by an amount \( F_0 \), the effect is

\[ \mathcal{F}^{-1}[X(f - F_0)] = e^{j2\pi f F_0} x(t) \]  

Note that the modulation Eq. (3.39) states that multiplying a signal \( x(t) \) by \( e^{j2\pi f_0 t} \) translates the spectrum of \( x(t) \) onto the frequency \( F_0 \), this is the frequency modulation (FM) principle.

**Differentiation and Integration**
Let \( x(t) \) be a continuous time signal with Fourier transform \( X(f) \).

\[ x(t) = \int_{-\infty}^{\infty} X(f) e^{j2\pi f t} df \]  

Then by differentiating both sides of the Fourier transform Eq. (3.40) we obtain

\[
\frac{dx(t)}{dt} = \int_{-\infty}^{\infty} j2\pi f X(f) e^{j2\pi f t} df \tag{3.41}
\]

That is multiplication of \(X(f)\) by the factor \(j2\pi f\) in the frequency domain is equivalent to differentiation of \(x(t)\) in time. Similarly division of \(X(f)\) by \(j2\pi f\) is equivalent to integration of the function of time \(x(t)\)

\[
\int_{-\infty}^{\infty} x(\tau) d\tau \leftrightarrow \frac{1}{j2\pi f} X(f) + \pi X(0) \delta(f) \tag{3.42}
\]

Where the impulse term on the right-hand side reflects the dc or average value that can result from the integration.

**Time and Frequency Scaling**

If \(x(t)\) and \(X(f)\) are Fourier transform pairs then

\[
x(\alpha t) \leftrightarrow \frac{1}{\alpha} X\left(\frac{f}{\alpha}\right) \tag{3.43}
\]

For example try to say something very slowly, then \(\alpha > 1\), your voice spectrum will be compressed and you may sound like a slowed down tape or disc, you can do the reverse and the spectrum would be expanded and your voice shifts to higher frequencies, This property is further illustrated in section 3.9.4.

**Convolution**

The convolution integral of two signals \(x(t)\) and \(h(t)\) is defined as

\[
y(t) = \int_{-\infty}^{\infty} x(\tau) h(t - \tau) d\tau \tag{3.44}
\]

The convolution integral is also written as

\[
y(t) = x(t) * h(t) \tag{3.45}
\]

where asterisk * denotes the convolution operation. The convolution integral is used to obtain the time-domain response of linear systems to arbitrary inputs as will be discussed in later sections.
Sec. 3.3 Fourier Transform

The Convolution Property of the Fourier Transform. It can be shown that convolution of two signals in the time domain corresponds to multiplication of the signals in the frequency domain, and conversely multiplication in the time domain corresponds to convolution in the frequency domain. To derive the convolutional property of the Fourier transform take the Fourier transform of the convolution of the signals \( x(t) \) and \( h(t) \) as

\[
\int_{-\infty}^{\infty} \left( \int_{-\infty}^{\infty} x(\tau) h(t-\tau) d\tau \right) e^{-j2\pi ft} dt = \int_{-\infty}^{\infty} x(\tau) e^{-j2\pi f\tau} d\tau \int_{-\infty}^{\infty} h(t-\tau) e^{-j2\pi f(t-\tau)} dt = X(f)H(f)
\]

(3.46)

Duality
Comparing the Fourier transform and the inverse Fourier transform relations we observe a symmetric relation between them. In fact the main difference between Eqs. (3.20) and (3.21) is a negative sign in the exponent of \( e^{-j2\pi f} \) in Eq (3.21). This symmetry leads to a property of Fourier transform known as the duality principle and stated as

\[
x(t) \xrightarrow{\mathcal{F}} X(f) \quad \text{and} \quad X(t) \xrightarrow{\mathcal{F}} x(f)
\]

(3.47)

As illustrated in Fig 3.11, the Fourier transform of a rectangular function of time \( r(t) \) has the form of a sinc pulse function of frequency \( \text{sinc}(f) \). From the duality principle the Fourier transform of a sinc function of time \( \text{sinc}(t) \) is a rectangular function of frequency \( R(f) \).

![Figure 3.11 Illustration of the principle of duality.](image-url)
Parseval's Theorem: Energy Relationship in Time and Frequency

Parseval’s relation states that the energy of a signal can be computed by integrating the squared magnitude of the signal either over the time domain or over the frequency domain. If \( x(t) \) and \( X(f) \) are a Fourier transform pair, then

\[
\text{Energy} = \int_{-\infty}^{\infty} |x(t)|^2 \, dt = \int_{-\infty}^{\infty} |X(f)|^2 \, df
\]  
(3.48)

This expression referred to as Parseval’s relation follows from a direct application of the Fourier transform.

Example 3.9 The Spectrum of a Finite Duration Signal

Find and sketch the frequency spectrum of the following finite duration signal

\[
x(t) = \sin(2\pi F_0 t) \quad -NT_0/2 \leq t \leq NT_0/2 \quad (3.49)
\]

where \( T_0=1/F_0 \) is the period and \( \omega_0=2\pi/T_0 \).

Solution: Substitute for \( x(t) \) and its non-zero valued limits in the Fourier transform Eq. (3.21)

\[
X(f) = \int_{-NT_0/2}^{NT_0/2} \sin(2\pi F_0 t)e^{-j2\pi ft} \, dt
\]  
(3.50)

substituting \( \sin(2\pi F_0 t) = (e^{j2\pi F_0 t} - e^{-j2\pi F_0 t})/2j \) in (3.50) gives

\[
X(f) = \int_{-NT_0/2}^{NT_0/2} \frac{e^{j2\pi F_0 t} - e^{-j2\pi F_0 t}}{2j} e^{-j2\pi ft} \, dt \\
= \int_{-NT_0/2}^{NT_0/2} \frac{e^{-j2\pi (f-F_0)t}}{2j} \, dt - \int_{-NT_0/2}^{NT_0/2} \frac{e^{-j2\pi (f+F_0)t}}{2j} \, dt
\]  
(3.51)

Evaluating the integrals yields
Sec. 3.3 Fourier Transform

\[ X(f) = \frac{e^{-j\pi(f-F_0)NT_0} - e^{j\pi(f-F_0)NT_0}}{4\pi(f-F_0)} - \frac{e^{-j\pi(f+F_0)NT_0} - e^{j\pi(f+F_0)NT_0}}{4\pi(f+F_0)} \]

\[ = \frac{-j}{2\pi(f-F_0)} \sin(\pi(f-F_0)NT_0) + \frac{j}{2\pi(f+F_0)} \sin(\pi(f+F_0)NT_0) \]

\[ X(f) = -jNT_0\sin(\pi(f - F_0)NT_0) + jNT_0\sin(\pi(f + F_0)NT_0) \] (3.53)

Matlab code for drawing the spectrum of a finite duration sinewave.

```matlab
% Sinewave Period T0, Number of Cycles in the window N
T0=.001; F0=1/T0; N=2000;
for Nc=1:4;
    for f=1:999
        if ((f-f0)~=0)
            x(Nc,f)=Nc*T0*sin(pi*(f-F0)*T0*Nc)/(pi*(f-F0)*T0*Nc);
        end
    end
    x(Nc,1000)=Nc*T0;
end
plot(x);
```

Figure 3.12 The spectrum of a finite duration sine wave with the increasing length of observation. \( N_c \) is the number of cycles in the observation window. Note that the width of the main lobe depends inversely on the duration of the signal. The signal spectrum can be expressed as the sum of two shifted sinc functions.

\[ X(f) = -jNT_0\sin(\pi(f - F_0)NT_0) + jNT_0\sin(\pi(f + F_0)NT_0) \] (3.53)

Note that energy of a finite duration sinewave is spread in frequency between the main lobe and the side lobes of the sinc function. Fig. 3.12 demonstrates that as the window length increases the energy becomes more concentrated in the main lobe and in the limit for an infinite duration window the spectrum tends to an impulse positioned at the frequency \( F_0 \).
Example 3.10 Calculation of the bandwidth for transmission of data at a rate of \( r_b \) bits per second. In its simplest form binary data can be represented as a sequence of amplitude modulated pulses as

\[
x(t) = \sum_{m=0}^{N-1} A(m) p(t - mT_b)
\]

where \( A(m) \) may be +1 or a –1 and

\[
p(t) = \begin{cases} 
1 & |t| \leq T_b / 2 \\
0 & |t| > 0
\end{cases}
\]

The Fourier transform of \( x(t) \) is given by

\[
X(f) = \sum_{m=0}^{N-1} A(m) P(f) e^{-j2\pi mfT_b} = P(f) \sum_{m=0}^{N-1} A(m) e^{-j2\pi mfT_b}
\]

The power spectrum of this signal is obtained as

\[
E[X(f)X^*(f)] = \left[ \sum_{m=-\infty}^{\infty} A(m) P(f) e^{-j2\pi mfT_b} \sum_{n=-\infty}^{\infty} A(n) P^*(f) e^{j2\pi nfT_b} \right] \\
= |P(f)|^2 \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} E[A(m)A(n)] e^{-j2\pi (m-n)fT_b}
\]

Now assuming that the data is uncorrelated we have

\[
E[A(m)A(n)] = \begin{cases} 
1 & m = n \\
0 & m \neq n
\end{cases}
\]

Substituting Equations (1) in (0) we have

\[
E[X(f)X^*(f)] = N |P(f)|^2
\]

From Equation (0) the bandwidth required from a sequence of pulses is basically the same as the bandwidth of a single pulse. From Figure (3.11) the main lobe width is \( 2r_b \).
3.3.3 Fourier Transform of a Sampled Signal

A sampled signal $x(m)$ can be modelled as

$$x(m) = \sum_{m=-\infty}^{\infty} x(t) \delta(t - mT_s)$$  \hspace{1cm} (3.54)

where $m$ is the discrete time variable and $T_s$ is the sampling period. The Fourier transform of $x(m)$, a sampled version of a continuous signal $x(t)$, can be obtained from Eq. (3.21) as

$$X(f) = \int_{-\infty}^{\infty} x(t) \delta(t - mT_s) e^{-j2\pi fm} dt = \sum_{m=-\infty}^{\infty} \int_{-\infty}^{\infty} x(t) \delta(t - mT_s) e^{-j2\pi fm} dt$$

$$= \sum_{m=-\infty}^{\infty} x(mT_s) e^{-j2\pi mf/F_s}$$  \hspace{1cm} (3.55)

For convenience of notation and without loss of generality it is often assumed that sampling frequency $F_s = 1/T_s = 1$, hence

$$X_s(f) = \sum_{m=-\infty}^{\infty} x(m) e^{-j2\pi mf}$$  \hspace{1cm} (3.56)

The inverse Fourier transform of a sampled signal is defined as

$$x(m) = \frac{1}{2} \int_{-1/2}^{1/2} X_s(f) e^{j2\pi fm} df$$  \hspace{1cm} (3.57)

Note that $x(m)$ and $X_s(f)$ are equivalent in that they contain the same information in different domains. In particular, as expressed by the Parseval's theorem, the energy of the signal may be computed either in the time or in the frequency domain

$$\text{Signal Energy} = \sum_{m=-\infty}^{\infty} x^2(m) = \frac{1}{2} \int_{-1/2}^{1/2} |X_s(f)|^2 df$$  \hspace{1cm} (3.58)

Example 3.10 Show that the spectrum of a sampled signal is periodic with a period equal to the sampling frequency $F_s$. 

Solution: substitute $f + kF_s$ for the frequency variable $f$ in Eq. (3.56)

$$X(f + kF_s) = \sum_{m=-\infty}^{\infty} x(mT_s) e^{-j 2\pi m (f + kF_s) / F_s} = \sum_{m=-\infty}^{\infty} x(mT_s) e^{-j 2\pi m f / F_s} e^{-j 2\pi m kF_s / F_s} = X(f)$$

(3.59)

Fig. 3.13.a shows the spectrum of a band-limited continuous-time signal. As shown in Fig. 3.13.b after the signal is sampled its spectrum becomes periodic.

![Figure 3.13](image)

**Figure 3.13** The spectrum of: (a) a continuous signal, and (b) its sampled version.

$$x(0) \quad x(1) \quad x(2) \quad \ldots \quad x(N-2) \quad x(N-1)$$

**Discrete Fourier Transform**

$$X(k) = \sum_{m=0}^{N-1} x(m) e^{-j \frac{2\pi kn}{N}}$$

![Figure 3.14](image)

**Figure 3.14** Illustration of the DFT as a parallel-input parallel-output signal processor.
3.4 Discrete Fourier Transform (DFT)

When a non-periodic signal is sampled, its Fourier transform becomes a periodic but continuous function of frequency, as shown in Eq. (3.59). The discrete Fourier transform (DFT) is derived from sampling the Fourier transform of a discrete-time signal. For a finite duration discrete-time signal \( x(m) \) of length \( N \) samples, the discrete Fourier transform (DFT) is defined as \( N \) uniformly spaced spectral samples

\[
X(k) = \sum_{m=0}^{N-1} x(m)e^{-\frac{2\pi j mk}{N}} \quad k = 0, \ldots, N-1 \tag{3.60}
\]

Comparing Eqs. (3.60) and (3.56) we see that the DFT consists of \( N \) equi-spaced samples taken from one period \((2\pi)\) of the continuous spectrum of the discrete time signal \( x(m) \). The inverse discrete Fourier transform (IDFT) is given by

\[
x(m) = \frac{1}{N} \sum_{k=0}^{N-1} X(k)e^{\frac{2\pi j mk}{N}} \quad m = 0, \ldots, N-1 \tag{3.61}
\]

A periodic signal has a discrete spectrum. Conversely any discrete frequency spectrum belongs to a periodic signal. Hence the implicit assumption in the DFT theory, is that the input signal \( x(m) \) is periodic with a period equal to the observation window length of \( N \) samples.

**Example:** Derivation of inverse discrete Fourier transform Obtain the inverse DFT from the DFT equation. The discrete Fourier transform (DFT) is given by

\[
X(k) = \sum_{m=0}^{N-1} x(m)e^{-\frac{j 2\pi km}{N}} \quad k = 0, \ldots, N-1
\]

Multiply both sides of the DFT equation by \( e^{j 2\pi kn/N} \) and take the summation as
Using the orthogonality principle, the inverse DFT equation can be derived as

\[ x(m) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{-\frac{j2\pi km}{N}} \]

3.4.1 Time and Frequency Resolutions: The Uncertainty Principle

Signals such as speech, music or image are composed of nonstationary — i.e. time-varying and/or space varying — events. For example speech is composed of a string of short-duration sounds called phonemes, and an image is composed of various objects. When using the DFT it is desirable to have a high enough time and space resolution in order to obtain the spectral characteristics of each individual elementary event or object in the input signal. However there is a fundamental trade-off between the length, i.e. the time or space resolution, of the input signal and the frequency resolution of the output spectrum. The DFT takes as the input a window of \( N \) uniformly spaced time domain samples \( x(0), x(1), \ldots, x(N-1) \) of duration \( \Delta T = N T_s \), and outputs \( N \) spectral samples \( X(0), X(1), \ldots, X(N-1) \) spaced uniformly between zero Hz and the sampling frequency \( F_s = 1/T_s \) Hz. Hence the frequency resolution of the DFT spectrum \( \Delta f \), i.e. the space between successive frequency samples, is given by

\[ \Delta f = \frac{1}{\Delta T} = \frac{1}{NT_s} = \frac{F_s}{N} \]  

Example 3.11 A DFT is used in a DSP system for the analysis of an analog signal with a frequency content of up to 10 kHz. Calculate: (i) the minimum sampling rate \( F_s \) required, and (ii) the number of samples required for the DFT to achieve a frequency resolution of 10 Hz at the minimum sampling rate.
Solution:
(i) Sampling rate $> 2 \times 10$ kHz, say 22 kHz, and

(ii) $\Delta f = \frac{F}{N}$

\[ 10 = \frac{22000}{N} \]

\[ N \geq 2200. \]

Example 3.12 Write a MATLAB program to explore the spectral resolution of a signal consisting of two sinewaves, closely spaced in frequency, with the varying length of the observation window.

Solution:
In the following program the two sinewaves have frequencies of 100 Hz and 110 Hz, and the sampling rate is 1 kHz. We experiment with two time windows of length $N_1=1024$ with a theoretical frequency resolution of $\Delta f=1000/1024=0.98$ Hz, and $N_2=64$ with a theoretical frequency resolution $\Delta f=1000/64=15.7$ Hz.

Fs=1000; F1=100; F2=110;
N=1:1024; N1=1024; N2=64;
x1=sin(2*pi*F1*N/Fs); x2=sin(2*pi*F2*N/Fs); y=x1+x2;

Figure 3.15 Illustration of time and frequency resolutions: (a) sum of two sinewaves with 10 Hz difference in their frequencies, (b) the spectrum of a segment of 64 samples from demonstrating insufficient frequency resolution to separate the sinewaves, (c) the spectrum of a segment of 1024 samples has sufficient resolution to show the two sinewaves.
Y1=abs(fft(y(1:N1))); Y2=abs(fft(y(1:N2))); figure(1); plot(y); figure(2); plot(Y1(1:N1/2)); figure(3); plot(Y2(1:N2/2));

3.4.2 The Effect of Finite Length Data on DFT (Windowing)

In a practical situation we have either with a short length signal, or with a long signal of which the DFT can only handle one segment at a time. Having a short segment of \( N \) samples of a signal or taking a slice of \( N \) samples from a signal is equivalent to multiplying the signal by a unit-amplitude rectangular pulse window of \( N \) samples. Therefore an \( N \)-sample segment of a signal \( x(m) \) is equivalent to

\[
x_w(m) = w(m)x(m) \quad (3.63)
\]

where \( w(m) \) is a rectangular pulse of \( N \) samples duration given is

\[
w(m) = \begin{cases} 1 & 0 \leq m \leq N - 1 \\ 0 & \text{otherwise} \end{cases} \quad (3.64)
\]

Multiplying two signals in time is equivalent to the convolution of their frequency spectra. Thus the spectrum of a short segment of a signal is convolved with the spectrum of a rectangular pulse as

\[
X_w(k) = W(k) \ast X(k) \quad (3.65)
\]

The result of this convolution is some spreading of the signal energy in the frequency domain as illustrated in the next example.

**Example 3.13** Find the DFT of a rectangular window given by

\[
w(m) = \begin{cases} 1 & 0 \leq m \leq N - 1 \\ 0 & \text{otherwise} \end{cases} \quad (3.66)
\]

**Solution:** Taking the DFT of \( w(m) \), and using the convergence formula for the partial sum of a geometric series, described in the appendix A, we have

\[
W(k) = \sum_{m=0}^{N-1} w(m)e^{-j\frac{2\pi}{N}mk} = \frac{1 - e^{-j2\pi k}}{1 - e^{-j\frac{2\pi}{N}k}} = e^{-j\frac{(N-1)\pi k}{N}} \frac{\sin(\pi k)}{\sin(\pi k / N)} \quad (3.67)
\]

Note that for the integer values of \( k \), \( w(k) \) is zero except for \( k=0 \).

**Example 3.14** Find the spectrum of an \( N \)-sample segment of a complex sinewave with a fundamental frequency \( F_0=1/T_0 \).
Solution: Taking the DFT of \( x(m) = e^{-j2\pi_0 m} \) we have

\[
X(k) = \sum_{m=0}^{N-1} e^{-j2\pi_0 m} \left( e^{-j2\pi k/N}ight)^m = \sum_{m=0}^{N-1} e^{-j2\pi (F_0 - k/N)m} \\
= \frac{1 - e^{-j2\pi(NF_0-k)}}{1 - e^{-j2\pi(NF_0-k)/N}} = e^{-\frac{(N-1)}{N}\pi(NF_0-k)} \frac{\sin(\pi(NF_0-k))}{\sin(\pi(NF_0-k)/N)}
\]

(3.68)

Note that for integer values of \( k \), \( X(k) \) is zero at all samples but one \( k=0 \).

3.4.3 End-Point Effects in DFT; Spectral Energy Leakage and Windowing

In DFT the input signal is assumed to be periodic, with a period equal to the length of the observation window of \( N \) samples. Now, for a sinusoidal input signal if there is an integer number of cycles within the observation window, as in Fig. 3.16.a, then the assumed periodic waveform is the same as an infinite length pure sinusoid. But if the observation window contains a non-integer number of cycles of a sinusoid then the assumed periodic waveform will not be a pure sine wave and will have end-point discontinuities. The spectrum of the signal then differs from the spectrum of sinewave as illustrated in Fig. 3.16.b. The overall effects of finite length window and end-point discontinuities are:

1. The spectral energy which could have been concentrated at a single point, or in a narrow band of frequencies is spread over a larger band of frequencies.
2. A smaller amplitude signal, located in frequency near a larger amplitude signal, may be obscured by one of the larger signal’s side-lobes. That is side-lobes of a large amplitude signal may interfere with the main-lobe of a nearby small amplitude signal.

The end-point problems may be alleviated using a window that gently drops to zero. One such window is a raised cosine window of the form

\[ w(m) = \begin{cases} 
\alpha - (1 - \alpha) \cos \frac{2\pi m}{N} & 0 \leq m \leq N - 1 \\
0 & \text{otherwise} 
\end{cases} \]  

(3.69)

For \( \alpha = 0.5 \) we have the Hanning window also known as the raised cosine window

\[ w_{\text{Han}}(m) = 0.5 - 0.5 \cos \frac{2\pi m}{N} \quad 0 \leq m \leq N - 1 \]  

(3.70)

For \( \alpha = 0.54 \) we have the Hamming window

\[ w_{\text{Ham}}(m) = 0.54 - 0.46 \cos \frac{2\pi m}{N} \quad 0 \leq m \leq N - 1 \]  

(3.71)

![Figure 3.17](image)

Figure 3.17 (a) Rectangular window frequency response, (b) Hamming window frequency response, (c) Hanning window frequency response.

### 3.4.3 Spectral Smoothing
The spectrum of a short length signal can be interpolated to obtain a smoother spectrum. Interpolation of the frequency spectrum \( X(k) \) is achieved by zero-padding of the time domain signal \( x(m) \). Consider a signal of length \( N \) samples \([x(0), \ldots, x(N-1)]\). Increase the signal length from \( N \) to \( 2N \) samples by padding \( N \) zeros to obtain the padded sequence \([x(0), \ldots, x(N-1), 0, \ldots, 0]\). The DFT of the padded signal is given by

\[
X(k) = \sum_{m=0}^{2N-1} x(m) e^{-j\frac{2\pi mk}{2N}} = \sum_{m=0}^{N-1} x(m) e^{-j\frac{2\pi mk}{N}}
\]

\( k = 0, \ldots, 2N-1 \) \hspace{1cm} (3.72)

The spectrum of the zero-padded signal, Eq. (3.72), is composed of \( 2N \) spectral samples; \( N \) of which, \([X(0), X(2), X(4), X(6), \ldots X(2N-2)]\) are the same as those that would be obtained from a DFT of the original \( N \) samples, and the other \( N \) samples \([X(1), X(3), X(5), X(6), \ldots X(2N-1)]\) are interpolated spectral lines that result from zero-padding. Note that zero padding does not increase the spectral resolution, it merely has an interpolating or smoothing effect in the frequency domain, as illustrated in Fig 3.18.

\[\text{Figure 3.18} \quad \text{Illustration of the interpolating effect, in the frequency domain, of zero padding a signal in the time domain.}\]
3.5 Short-Time Fourier Transform

In Fourier transform it is assumed that the signal is stationary, meaning the signal statistics, such as the mean, the power, and the power spectrum, are time-invariant. Most real life signals such as speech, music and image signals are nonstationary in that their amplitude, power, spectral composition and other features changes continuously with time. To apply Fourier transform to nonstationary signals the signal is divided into appropriately short-time windows, such that within each window the signal can be assumed to be time-invariant. The Fourier transform applied to the short signal segment within each window is known as the short-time Fourier Transform. Fig. 3.19 illustrates the segmentation of a speech signal into a sequence of overlapping, hamming windowed, short segments. The choice of window length is a compromise between the time resolution and the frequency resolution. For Audio signals a time window of about 25 ms, corresponding to a frequency resolution of 40 Hz is normally adopted.
Fourier Analysis and Wavelet Analysis

James S. Walker

In this article we will compare the classical methods of Fourier analysis with the newer methods of wavelet analysis. Given a signal, say a sound or an image, Fourier analysis easily calculates the frequencies and the amplitudes of those frequencies which make up the signal. This provides a broad overview of the characteristics of the signal, which is important for theoretical considerations. However, although Fourier inversion is possible under certain circumstances, Fourier methods are not always a good tool to recapture the signal, particularly if it is highly nonsmooth: too much Fourier information is needed to reconstruct the signal locally. In these cases, wavelet analysis is often very effective because it provides a simple approach for dealing with local aspects of a signal. Wavelet analysis also provides us with new methods for removing noise from signals that complement the classical methods of Fourier analysis. These two methodologies are major elements in a powerful set of tools for theoretical and applied analysis.

This article contains many graphs of discrete signals. These graphs were created by the computer program FAWAV, A Fourier–Wavelet Analyzer, being developed by the author.

James S. Walker is professor of mathematics at the University of Wisconsin-Eau Claire. His e-mail address is walkerjs@uwec.edu.

The author would like to thank Hugo Rossi, Steven Krantz, his colleague Marc Goulet, and two anonymous reviewers for their helpful comments during the writing of this article.

Frequency Information, Denoising

As an example of the importance of frequency information, we will examine how Fourier analysis can be used for removing noise from signals. Consider a signal \( f(x) \) defined over the unit interval (where here \( x \) stands for time). The period 1 Fourier series expansion of \( f \) is defined by \( \sum_{n \in \mathbb{Z}} c_n e^{i2\pi nx} \), with \( c_n = \int_0^1 f(x) e^{-i2\pi nx} \, dx \). Each Fourier coefficient, \( c_n \), is an amplitude associated with the frequency \( n \) of the exponential \( e^{i2\pi nx} \). Although each of these exponentials has a precise frequency, they all suffer from a complete absence of time localization in that their magnitudes, \( |e^{i2\pi nx}| \), equal 1 for all time \( x \).

To see the importance of frequency information, let us examine a problem in noise removal. In Figure 1(a)[top] we show the graph of the signal

\[
(1) \quad f(x) = \begin{cases} 
5 \cos 2\pi \nu x & \left| e^{-640\pi(x-1/8)^2} + e^{-640\pi(x-3/8)^2} + e^{-640\pi(x-4/8)^2} + e^{-640\pi(x-6/8)^2} + e^{-640\pi(x-7/8)^2} \right.
\end{cases}
\]

where the frequency, \( \nu \), of the cosine factor is 280. Such a signal might be used by a modem for transmitting the bit sequence 1 0 1 1 0 1 1. The Fourier coefficients for this signal are shown in Figure 1(b)[top]. The highest magnitude coefficients are concentrated around the frequencies \( \pm 280 \). Suppose that when this signal is received, it is severely distorted by added noise: see Figure 1(a)[middle]. Using Fourier analysis, we can remove most of this noise. Computing the noisy signal’s Fourier coefficients, we obtain the graph shown in Figure 1(b)[middle]. The original signal’s largest magnitude Fourier coefficients are clus-
tered around the frequency positions ±280. The Fourier coefficients of the added noise are localized around the origin, and they decrease in magnitude until they are essentially zero near the frequencies ±280. Thus, the original signal’s coefficients and the noise’s coefficients are well separated. To remove the noise from the signal, we multiply the noisy signal’s coefficients by a filter function, which is 1 where the signal’s coefficients are concentrated and 0 where the noise’s coefficients are concentrated. We then recover essentially all of the signal’s coefficients; see Figure 1(b)[bottom]. Performing a Fourier series partial sum with these recovered coefficients, we obtain the denoised signal, which is shown in Figure 1(a)[bottom]. Clearly, the bit sequence 1011011 can now be determined from the denoised signal, and the denoised signal is a close match of the original signal. In the section “Signal Denoising” we shall look at another example of this method and also discuss how wavelets can be used for noise removal.

Signal Compression

As the example above shows, Fourier analysis is very effective in problems dealing with frequency location. However, it is often very ineffective at representing functions. In particular, there are severe problems with trying to analyze transient signals using classical Fourier methods. For example, in Figure 2(a)[top] we show a discrete signal obtained from \( M = 1024 \) values \( f_j = F(j/M) \), \( j = 0, \ldots, M-1 \) of the function \( F(x) = e^{-10^5 \pi (x - 0.6)^2} \). For this example, we compute the discrete Fourier series coefficients \( \hat{f}_n \) defined by

\[
\hat{f}_n = M^{-1} \sum_{j=0}^{M-1} f_j e^{-i2\pi nj/M}
\]

for \( n = -\frac{1}{2} M + 1, \ldots, 0, \ldots, \frac{1}{2} M \). The discrete Fourier coefficients \( \hat{f}_n \) can be calculated by a fast Fourier transform (FFT) algorithm and are the discrete analog of the Fourier coefficients \( c_n \) for \( F \), when \( f_j = F(j/M) \). Moreover, \( \hat{f}_n \) is just a Riemann sum approximation of the integral that defines \( c_n \).

The magnitudes of the discrete Fourier coefficients for this transient damp down to zero very slowly (their graph is a very wide bell-shaped curve with maximum at the origin). Consequently, to represent the transient well, one must retain most if not all of these Fourier coefficients. In Figure 2(a)[bottom] we show the results obtained from trying to compress the transient by computing a discrete partial sum \( \sum_{n=-104}^{104} \hat{f}_n e^{i2\pi nj/M} \) using only one-fifth of the Fourier coefficients. Clearly, even a moderate compression ratio of 5:1 is not effective.

Wavelets, however, are often very effective at representing transients. This is because they are designed to capture information over a large range of scales. A wavelet series expansion of a function \( f \) is defined by

\[
\sum_{n,k \in \mathbb{Z}} \beta^n_k 2^{n/2} \psi(2^n x - k)
\]

with

\[
\beta^n_k = \int_{-\infty}^{\infty} f(x) 2^{n/2} \psi(2^n x - k) \, dx.
\]

The function \( \psi(x) \) is called the wavelet, and the coefficients \( \beta^n_k \) are called the wavelet coefficients. The function \( 2^{n/2} \psi(2^n x - k) \) is the

---

2. The sum \( \sum_{n=-512}^{512} \hat{f}_n e^{i2\pi nj/M} \), which uses all of the discrete Fourier coefficients, equals \( f_j \).
wavelet shrunk by a factor of $2^n$ if $n$ is positive (magnified by a factor of $2^{-n}$ if $n$ is negative) and shifted by $k2^{-n}$ units. The factor $2^{n/2}$ in the expression $2^{n/2} \psi(2^n x - k)$ preserves the $L^2$-norm.

Since the wavelet series depends on two parameters of scale and translation, it can often be very effective in analyzing signals. These parameters make it possible to analyze a signal's behavior at a dense set of time locations and with respect to a vast range of scales, thus providing the ability to zoom in on the transient behavior of the signal. For example, let us examine the earlier transient using a discretized version of a wavelet series. We shall use a Daubechies order 4 wavelet (Daub4 for short; see the section "Daubechies Wavelets"). In Figure 2(b) we show all of the 1024 wavelet coefficients of this transient and observe that most of these coefficients are close to 0 in magnitude. Consequently, by retaining only the largest magnitude coefficients for use in a wavelet series, we obtain significant compression. In Figure 2(b) we show the reconstruction of the transient using only the top 4% in magnitude of the wavelet coefficients, a 25:1 compression ratio. Notice how accurately the transient is represented. In fact, the maximum error at all computed points is less than $9.95 \times 10^{-14}$. There is an important application here to the field of signal transmission. By transmitting only these 4% of the wavelet coefficients, the information in the signal can be transmitted 25 times faster than if we transmitted all of the original signal. This provides a considerable boost in efficiency of transmission.

We shall look at more examples of compression in the section "Compression of Signals", but first we shall describe how wavelet analysis works.

The Haar Wavelet

In order to understand how wavelet analysis works, it is best to begin with the simplest wavelet, the Haar wavelet. Let $1_A(x)$ denote the indicator function of the set $A$, defined by $1_A(x) = 1$ if $x \in A$ and $1_A(x) = 0$ if $x \notin A$. The Haar wavelet $\psi$ is defined by $\psi(x) = 1_{[0,1]}(x) - 1_{[\frac{1}{2},1]}(x)$. It is 0 outside of $[0,1)$, so it is well localized in time, and it satisfies

$$\int_{-\infty}^{\infty} \psi(x) \, dx = 0, \quad \int_{-\infty}^{\infty} |\psi(x)|^2 \, dx = 1.$$

The Haar wavelet $\psi(x)$ is closely related to the function $\phi(x)$ defined by $\phi(x) = 1_{[0,1]}(x)$. This function $\phi(x)$ is called the Haar scaling function. Clearly, the Haar wavelet and scaling function satisfy the identities

$$\psi(x) = \phi(2x) - \phi(2x-1),$$
$$\phi(x) = \phi(2x) + \phi(2x-1),$$

and the scaling function satisfies

$$\int_{-\infty}^{\infty} \phi(x) \, dx = 1, \quad \int_{-\infty}^{\infty} |\phi(x)|^2 \, dx = 1.$$

The Haar wavelet $\psi(x)$ generates the system of functions $\{2^{n/2} \psi(2^n x - k)\}$. It is possible to show directly that $\{2^{n/2} \psi(2^n x - k)\}$ is an orthonormal basis for $L^2(\mathbb{R})$, but it is more illuminating to put the discussion on an axiomatic level. This axiomatic approach leads to the Daubechies wavelets and many other wavelets as well. We begin by defining the subspaces $\{V_n\}_{n \in \mathbb{Z}}$ of $L^2(\mathbb{R})$ in the following way:
Let \( V_n = \left\{ \text{step functions in } L^2(\mathbb{R}), \text{ constant} \right\} \) on the intervals \( \left[ \frac{k}{2^n}, \frac{k+1}{2^n} \right), k \in \mathbb{Z} \). This set of subspaces \( \{V_n\}_{n \in \mathbb{Z}} \) satisfies the following five axioms [6]:

**Axioms for a Multi-Resolution Analysis (MRA)**

- **Scaling:** \( f(x) \in V_n \) if and only if \( f(2x) \in V_{n+1} \).
- **Inclusion:** \( V_n \subset V_{n+1} \), for each \( n \).
- **Density:** closure of \( \left\{ \bigcup_{n \in \mathbb{Z}} V_n \right\} = L^2(\mathbb{R}) \).
- **Maximality:** \( \bigcap_{n \in \mathbb{Z}} V_n = \{0\} \).
- **Basis:** \( \exists \phi(x) \) such that \( \{\phi(x-k)\}_{k \in \mathbb{Z}} \) is an orthonormal basis for \( V_0 \).

To satisfy the basis axiom, we shall use the Haar scaling function \( \phi \) defined above. Then, by combining the scaling axiom with the basis axiom, we find that \( \{2^{n/2}\phi(2^n x - k)\}_{k \in \mathbb{Z}} \) is an orthonormal basis for \( V_n \). But the totality of all these orthonormal bases, consisting of the set \( \{2^{n/2}\phi(2^n x - k)\}_{k,n \in \mathbb{Z}} \), is not an orthonormal basis for \( L^2(\mathbb{R}) \) because the spaces \( V_n \) are not mutually orthogonal. To remedy this difficulty, we need what are called *wavelet subspaces*. Define the wavelet subspace \( W_n \) to be the orthogonal complement of \( V_n \) in \( V_{n+1} \). That is, \( W_n \) satisfies the equation \( V_{n+1} = V_n \oplus W_n \) where \( \oplus \) denotes the sum of mutually orthogonal subspaces. From the density axiom and repeated application of the last equation, we obtain \( L^2(\mathbb{R}) = V_0 \oplus \bigoplus_{n=0}^{\infty} W_n \). Decomposing \( V_0 \) in a similar way, we obtain \( L^2(\mathbb{R}) = \bigoplus_{n \in \mathbb{Z}} W_n \). Thus, \( L^2(\mathbb{R}) \) is an orthogonal sum of the wavelet subspaces \( W_n \).

Using (2) and the MRA axioms, it is easy to prove the following lemma.

**Lemma 1.** The functions \( \{\tilde{\psi}(x-k)\}_{k \in \mathbb{Z}} \) are an orthonormal basis for the subspace \( W_0 \).

It follows from the scaling axiom that \( \{2^{n/2}\psi(2^n x - k)\}_{k,n \in \mathbb{Z}} \) is an orthonormal basis for \( W_n \). Therefore, since \( L^2(\mathbb{R}) \) is the orthogonal sum of all the wavelet subspaces \( W_n \), we have obtained the following result.

**Theorem 1.** The functions

\[
\{2^{n/2}\psi(2^n x - k)\}_{k,n \in \mathbb{Z}}
\]

are an orthonormal basis for \( L^2(\mathbb{R}) \).

This orthonormal basis is the *Haar basis* for \( L^2(\mathbb{R}) \). There is also a Haar basis for \( L^2([0,1]) \). To obtain it, we first define *periodic wavelets* \( \tilde{\psi}_{n,k} \) by

\[
\tilde{\psi}_{n,k}(x) = \sum_{j \in \mathbb{Z}} 2^{n/2} \psi(2^n(x + j) - k).
\]

Note that these wavelets have period 1. Furthermore, \( \tilde{\psi}_{n,k} = 0 \) for \( n < 0 \), and \( \tilde{\psi}_{n,k+2^n} = \tilde{\psi}_{n,k} \) for all \( k \in \mathbb{Z} \) and \( n \geq 0 \). On the interval \([0,1]\), the periodic Haar wavelets \( \tilde{\psi}_{n,k} \) satisfy

\[
\tilde{\psi}_{n,k}(x) = 2^{n/2}\psi(2^n x - k)
\]

for \( n \geq 0 \) and \( k = 0, 1, \ldots, 2^n - 1 \). So we have the following theorem as a consequence of Theorem 1.

**Theorem 2.** The functions 1 and \( \tilde{\psi}_{n,k} \) for \( n \geq 0 \) and \( k = 0, 1, \ldots, 2^n - 1 \) are an orthonormal basis for \( L^2[0,1]\).

**Remark.** In the section “Daubechies Wavelets” we will make use of *periodized scaling functions*, \( \tilde{\phi}_{n,k} \) defined by

\[
\tilde{\phi}_{n,k}(x) = \sum_{j \in \mathbb{Z}} 2^{n/2} \phi(2^n(x + j) - k)
\]

for \( n \geq 0 \) and \( k = 0, 1, \ldots, 2^n - 1 \).

**Fast Haar Transform**

The relation between the Haar scaling function \( \phi \) and wavelet \( \psi \) leads to a beautiful set of relations between their coefficients as bases. Let \( \{\alpha_{n,k}^0\} \) and \( \{\beta_{n,k}^0\} \) be defined by

\[
\alpha_{n,k}^0 = \int_{-\infty}^{\infty} f(x) 2^{n/2} \phi(2^n x - k) \, dx,
\]

\[
\beta_{n,k}^0 = \int_{-\infty}^{\infty} f(x) 2^{n/2} \psi(2^n x - k) \, dx.
\]

Substituting \( 2^n x \) in place of \( x \) in the identities in (2), we obtain

\[
2^{n/2} \phi(2^n x) = \frac{1}{\sqrt{2}} [2^{n+1} \phi(2^{n+1} x)] + \frac{1}{\sqrt{2}} [2^{n+1} \phi(2^{n+1} x - 1)]
\]

\[
2^{n/2} \psi(2^n x) = \frac{1}{\sqrt{2}} [2^{n+1} \psi(2^{n+1} x)] - \frac{1}{\sqrt{2}} [2^{n+1} \psi(2^{n+1} x - 1)].
\]

It then follows that

\[
\alpha_{n,k}^0 = \frac{1}{\sqrt{2}} \alpha_{n+1,k} + \frac{1}{\sqrt{2}} \alpha_{n+1,k+1},
\]

\[
\beta_{n,k}^0 = \frac{1}{\sqrt{2}} \beta_{n+1,k} - \frac{1}{\sqrt{2}} \beta_{n+1,k+1}.
\]

This result shows that the \( n \)th level coefficients \( \alpha_{n,k}^0 \) and \( \beta_{n,k}^0 \) are obtained from the \( (n+1) \)st level coefficients \( \alpha_{n+1,k}^0 \) through multiplication by the following orthogonal matrix:
divide the number of components by 2. At the 
operations continue until we can no longer 
the 
unit interval. If we shrink the Haar scaling func-
matrix sums are applied 
ations just described are repeated, only now 
we apply an 
To sort the coefficients properly into two groups, 
the matrices used are 
operate only on the 
PM 
permutation matrix 
M/2 times and \( \mathcal{O} \) is the 
matrix defined in (7). Then, by applying the co-
efficient relations in (6) and using the fact that 
\( f_k = \alpha_k^R \), we obtain 
\[
\begin{align*}
H_M \{f_0, f_1, \ldots, f_{M-1}\}^T &= \left[ \alpha_0^{R-1}, \beta_0^{R-1}, \alpha_1^{R-1}, \beta_1^{R-1}, \\
&\quad \ldots, \alpha_{M-1}^{R-1}, \beta_{M-1}^{R-1}\right]^T.
\end{align*}
\]
To sort the coefficients properly into two groups, we apply an \( M \times M \) permutation matrix \( P_M \) as follows:
\[
P_M \left[ \alpha_0^{R-1}, \beta_0^{R-1}, \alpha_1^{R-1}, \beta_1^{R-1}, \\
&\quad \ldots, \alpha_{M-1}^{R-1}, \beta_{M-1}^{R-1}\right]^T = \left[ \alpha_0^{R-1}, \ldots, \alpha_{M-1}^{R-1}, \beta_0^{R-1}, \ldots, \beta_{M-1}^{R-1}\right]^T.
\]
If we go to the next lower level, the transfor-
mations just described are repeated, only now 
the matrices used are \( H_{M/2} \) and \( P_{M/2} \), and they operate only on the \( M/2 \)-length vector 
\{ \( \alpha_0^{R-1}, \ldots, \alpha_{M-1}^{R-1} \) \} to obtain the next level wavelet coeffi-
cients \{ \( \beta_0^{R-2}, \ldots, \beta_{M-1}^{R-2} \) \} and scaling coefficients \{ \( \alpha_0^{R-2}, \ldots, \alpha_{M-1}^{R-2} \) \}. These 
operations continue until we can no longer 
divide the number of components by 2. At the 
\( R = \log_2 M \) step, we obtain a single scale coeffi-
cient \( \alpha_0^R \) and a single wavelet coefficient \( \beta_0^R \), 
and at this last step the permutation \( P_2 \) is un-
necessary. The complete transformation, de-
noted by \( \mathcal{H} \), satisfies 
\[
\mathcal{H} = (H_2 \oplus I_{M-2})
\]
\[
\ldots (P_{M/2} \oplus I_{M/2}) (H_{M/2} \oplus I_{M/2}) P_M H_M
\]
where \( I_N \) is the \( N \times N \) identity matrix.

These matrix multiplications can be per-
formed rapidly on a computer. Multiplying by 
\( H_M \) requires only \( O(M) \) operations, since \( H_M \) 
consists mostly of zeroes. Similarly, the per-
mutation \( P_M \) requires \( O(M) \) operations. There-
fore, the whole transformation requires 
\( O(M) + O(M/2) + \ldots + O(2) = O(M) \) operations.
The transformation \( \mathcal{H} \) is called a fast Haar 
transform. It should be noted that FFTs, which 
have revolutionized scientific practice during 
the last thirty years, are \( O(M \log M) \) algorithms.

Since each \( H_k \) is an orthogonal matrix, and 
so is every permutation \( P_k \), it follows that \( \mathcal{H} \) is 
invertible. Its inverse is
\[
\mathcal{H}^{-1} = H_M^T P_M^T (H_{M/2}^T \oplus I_{M/2}) (P_{M/2}^T \oplus I_{M/2})
\]
\[
\ldots (H_2^T \oplus I_{M-2}).
\]
Therefore, the inverse operation is also an \( O(M) \) 
operation.

**Discrete Haar Series**

The fast Haar transform can be used for com-
puting partial sums of the discretized version 
of the following Haar wavelet series in \( L^2[0,1] \):
\[
\alpha_0^R + \sum_{n=0}^{2^R-1} \sum_{k=0}^{\infty} \beta_k^n \psi(2^R x - k).
\]
Let us assume, as in the previous section, that 
we have a discrete signal \( \{f_j\}_{j=0}^{M-1} \) associated 
with the time values \( \{x_j = j/M\}_{j=0}^{M-1} \) on the unit 
interval. Substituting these time values into (8) 
and restricting the upper limit of \( n \), we obtain 
\[
f_j = \alpha_0^R C_M + \sum_{n=0}^{R-1} \sum_{k=0}^{2^n-1} \beta_k^n C_M \psi(2^n x_j - k).
\]
The right side of this equation is just the transfor-
mation \( \mathcal{H}^{-1} \mathcal{H} \) applied to \( \{f_j\} \). The first 
part of this transformation, \( \mathcal{H} \{f_j\} \), produces 
the coefficients \( \alpha_0^R, \beta_0^R, \beta_1^R, \ldots, \beta_{M-1}^R \), and the 
second part, the application of \( \mathcal{H}^{-1} \), re-
produces the original data \( \{f_j\} \). The constant 
\( C_M \) is a scale factor which ensures that the 
constant vector \( C_M \) and the vectors \( \{C_M 2^{n/2} \psi(2^n x_j - k)\} \) 
are unit vectors in \( \mathbb{R}^M \) using the standard inner 
product. Consequently, \( C_M = \sqrt{1/M} \).

There are many ways of forming partial sums 
of discretized Haar series. The simplest ones con-
sist of multiplying the data by $H$, then setting some of the resulting coefficients equal to 0, and then multiplying by $H^{-1}$. A widely used method involves specifying a threshold. All coefficients whose magnitudes lie below this threshold are set equal to 0. This method is frequently used for noise removal, where coefficients whose magnitudes are significant only because of the added noise will often lie below a well-chosen threshold. We shall give an example of this in the section “Signal Denoising”. A second method keeps only the largest magnitude coefficients, while setting the rest equal to 0. This method is convenient for making comparisons when it is known in advance how many terms are needed. We used it in the compression example in the section “Signal Compression”. A third method, which we shall call the energy method, involves specifying a fraction of the signal’s energy, where the energy is the square root of the sum of the squares of the coefficients. We then retain the least number of the largest magnitude coefficients whose energy exceeds this fraction of the signal’s energy and set all other coefficients equal to 0. The energy method is useful for theoretical purposes: it is clearly helpful to be able to specify in advance what fraction of the signal’s energy is contained in a partial sum. We shall use the energy method frequently below.

Let us look at an example. Suppose our signal is

$$\{f_j = F(j/8192)\}_{j=0}^{8191}$$

where $F(x) = x1_{[0,5]}(x) + (x - 1)1_{[5,1)}(x)$. In Figure 3(a)[top] we show a Haar series partial sum, created by the energy method, which contains 99.5% of the energy of this signal. This partial sum, which used 229 coefficients out of a possible 8192, provides an acceptable visual representation of the signal. In fact, the sum of the squares of the errors is $2 \times 10^{-3}$. By comparison a 229 coefficient Fourier series partial sum suffers from serious drawbacks (see Figure 3(b)[top]). The sum of the squares of the errors is $4.5 \times 10^{-1}$, and there is severe oscillation and a Gibbs’ effect near $x = 0.5$. Although these latter two defects could be ameliorated using other summation methods ([11], Ch. 4), there would still be a significant deviation from the original signal (especially near $x = 0.5$).

This example illustrates how wavelet analysis hones right in on regions of high variability of signals and that Fourier methods try to smooth them out. The size of a function’s Fourier coefficients is related to the frequency content of the function, which is measured by integration of the function against completely unlocalized basis functions. For a function having a discontinuity, or some type of transient behavior, this produces Fourier coefficients that decrease in magnitude at a very slow rate. Consequently, a large number of Fourier coefficients are needed to accurately represent such signals. Wavelet series, however, use compactly supported basis functions which, at increasing levels of resolution, have rapidly decreasing supports and can zoom in on transient behavior. The transient behaviors contribute to the magnitude of only a small portion of the wavelet coefficients.

---

3The Haar transform is orthogonal, so it makes sense to specify energy in this way.

4In recent years, though, significant improvements have been achieved using local cosine bases [3, 7, 9, 5, 1].
Consequently, a small number of wavelet coefficients are needed to accurately represent such signals.

The Haar system performs well when the signal is constant over long stretches. This is because the Haar wavelet is supported on \([0, 1]\) and satisfies a 0th order moment condition, \(\int_{-\infty}^{\infty} \psi(x) \, dx = 0\). Therefore, if the signal \(f_j\) is constant over an interval \(a \leq x_j < b\) such that \([k2^{-n}, (k+1)2^{-n}) \subset [a, b]\), then the wavelet coefficient \(\beta^n_k\) equals 0. For example, suppose \(f_j = F(j/8192)\),

\[
F(x) = (8x - 1)1_{[.125,.25]}(x) + 1_{[.25,.75]}(x) + (7 - 8x)1_{[.75,.875]}(x).
\]

In Figure 3(a) we show a Haar series partial sum which contains 99.5% of the energy of this signal and uses only 92 coefficients out of a possible 8192. The fact that the signal is constant over three large subintervals of \([0, 1]\) accounts for the excellent compression in this example. In order to obtain wavelet bases that provide considerably more compression, we need a compactly supported wavelet \(\psi(x)\) which has more moments equal to zero. That is, we want

\[
\int_{-\infty}^{\infty} x^j \psi(x) \, dx = 0, \text{ for } j = 0, 1, \ldots, L - 1
\]

for an integer \(L \geq 2\). We say that such a wavelet has its first \(L\) moments equal to zero. For example, a Daub4 wavelet has its first 2 moments equal to zero. Using a Daub4 wavelet series for the signal above, it is possible to capture 99.5% of the energy using only 22 coefficients! See Figure 3(b). This improvement in compression is due to the fact that the Daub4 wavelet is supported on \([0, 3]\) and satisfies \(\int_{-\infty}^{\infty} \psi(x) \, dx = 0\) and \(\int_{-\infty}^{\infty} x^j \psi(x) \, dx = 0\). Consequently, if the signal is constant or linear over an interval \([a, b]\) which contains \([k2^{-n}, (k+1)2^{-n}) \subset [a, b]\), then the wavelet coefficient \(\beta^n_k\) will equal 0. In Figure 4(a) we show graphs of the magnitudes of the highest level Haar coefficients and Daub4 coefficients. Each magnitude \(|\beta^n_k|\) is plotted at the \(x\)-coordinate \(k2^{-12}\) for \(k = 0, 1, \ldots, 2^{12} - 1\). These graphs show that the highest level Haar coefficients are near 0 over the constant parts of \(F\), while the highest level Daub4 coefficients are near 0 over the constant and linear parts of \(F\). In Figure 4(b) we show graphs of the sums of the squares of all the coefficients, which show that almost all the Daub4 coefficients are near 0 over the constant and linear parts of \(F\), while the Haar coefficients are near 0 only over the constant parts of \(F\). Furthermore, the largest magnitude Daub4 coefficients are concentrated around the locations of the points of nondifferentiability of \(F\). This kind of local analysis illustrates one of the powerful features of wavelet analysis.

Looking again at Figure 3(b), we see that the most serious defects of the Daub4 compressed signal are near the points where \(F\) is non-differentiable. If, however, we consider the interval \([0.4, 0.6]\) where \(F\) is constant, the Daub4 compressed signal values differ from the values of \(F\) by no more than \(1.2 \times 10^{-15}\) at all of the 1641 discrete values of \(x\) in this interval. In contrast, a Fourier series partial sum using 23 coefficients differs by more than \(10^{-3}\) at 1441 of these 1641 values of \(x\). The Fourier series partial sum exhibits oscillations of amplitude \(6.5 \times 10^{-3}\) around the value 1 over this subin-
Because the Fourier coefficients for $F$ are only $O(n^{-2})$, using just the first 23 coefficients produces an oscillatory approximation to $F$ over all of $[0, 1)$, including the subinterval $[0, 0.6]$. The highest magnitude wavelet coefficients, however, are concentrated at the corner points for $F$, and their terms affect only a small portion of the partial sum (since their basis functions are compactly supported). Consequently, the wavelet series provides an extremely close approximation of $F$ over the subinterval $[0, 0.6]$.

A major defect of the Haar wavelet is its discontinuity. For one thing, it is unsatisfying to use discontinuous functions to approximate continuous ones. Even with discrete signals there can be undesirable jumps in Haar series partial sum values (see Figure 3(a)[bottom]). Therefore, we want to have a wavelet that is continuous. In the next section we will describe the Daubechies wavelets, which have their first $L \geq 2$ moments equal to zero and are continuous.

Daubechies Wavelets

It is possible to generalize the construction of the Haar wavelet so as to obtain a continuous scaling function $\phi(x)$ and a continuous wavelet $\psi(x)$. Moreover, Daubechies has shown how to make them compactly supported. We will briefly sketch the main ideas; more details can be found in [4, 5, 8, 10].

Generalizing from the case of the Haar wavelets, we require that $\phi(x)$ and $\psi(x)$ satisfy

$$\int_{-\infty}^{\infty} \phi(x) \, dx = 1, \quad \int_{-\infty}^{\infty} |\phi(x)|^2 \, dx = 1, \quad \int_{-\infty}^{\infty} |\psi(x)|^2 \, dx = 1. \quad (10)$$

The MRA axioms tell us that $\phi(x)$ must generate a subspace $V_0$ and that $V_0 \subset V_1$. Therefore,

$$\phi(x) = \sum_{k \in \mathbb{Z}} c_k \sqrt{2} \phi(2x - k) \quad \text{for some constants } \{c_k\}. \quad (11)$$

A simple proof, based on the MRA axioms, that $\{\psi(x - k)\}$ spans $W_0$ can be found in [10].

Equations (11) and (12) generalize the equations in (2) for the Haar case. The orthogonality of $\phi$ and $\psi$ leads to the following equation

$$\sum_{k \in \mathbb{Z}} (\sqrt{2})^k c_k = 0, \quad \sum_{k \in \mathbb{Z}} |c_k|^2 = 1. \quad (13)$$

This equation, and the second equation in (14) below, imply the orthogonality of the matrices, $W_N$, used in the fast wavelet transform which we shall discuss later in this section.

Combining (11) with the first two integrals in (10), it follows that

$$\sum_{k \in \mathbb{Z}} c_k = \sqrt{2}, \quad \sum_{k \in \mathbb{Z}} |c_k|^2 = 1. \quad (14)$$

Similarly, assuming that $L = 2$, the equations in (9) combined with (12) imply

$$\sum_{k \in \mathbb{Z}} (-1)^k c_k = 0, \quad \sum_{k \in \mathbb{Z}} k(-1)^k c_k = 0. \quad (15)$$

A more detailed treatment of the Daubechies wavelets can be found in [8].

Figure 5. (a) Signal. (b) 37-term Daub4 approximation. (c) 257-term Fourier cosine series approximation. (d) Highest level Daub4 wavelet coefficients of signal.
And, for \( L > 2 \), equations (9) and (12) yield additional equations similar to the ones in (15). There is a finite set of coefficients that solves the equations in (14) and (15), namely,

\[
\begin{align*}
c_0 &= \frac{1 + \sqrt{3}}{4\sqrt{2}}, & c_1 &= \frac{3 + \sqrt{3}}{4\sqrt{2}}, \\
c_2 &= \frac{3 - \sqrt{3}}{4\sqrt{2}}, & c_3 &= \frac{1 - \sqrt{3}}{4\sqrt{2}}
\end{align*}
\]

(16)

with all other \( c_k = 0 \). Using these values of \( c_k \), the following iterative solution of (11)

\[
\phi_0(x) = 1_{[0,1]}(x),
\]

(17) \( \phi_n(x) = \sum_{k \in \mathbb{Z}} c_k \sqrt{2} \phi_{n-1}(2x - k) \)

for \( n \geq 1 \), converges to a continuous function \( \phi(x) \) supported on \([0, 3]\). It then follows from (12) that the wavelet \( \psi(x) \) is also continuous and compactly supported on \([0, 3]\). This wavelet \( \psi \) we have been referring to as the Daubechies wavelet. The set of coefficients \( \{c_k\} \) in (16) is the smallest set of coefficients that produce a continuous compactly supported scaling function. Other sets of coefficients, related to higher values of \( L \), are given in [4] and [12].

Once the scaling function \( \phi(x) \) and the wavelet function \( \psi(x) \) have been found, then we proceed as we did above in the Haar case. We define the coefficients \( \{\alpha_n^0\} \) and \( \{\beta_n^0\} \) by the equations in (5), where now \( \phi \) and \( \psi \) are the Daubechies scaling function and wavelet, respectively. The scaling identity (11) and the wavelet definition (12) yield the following coefficient relations:

\[
\begin{align*}
\alpha_k^n &= \sum_{m \in \mathbb{Z}} c_m \alpha_{m+2k}^{n+1}, \\
\beta_k^n &= \sum_{m \in \mathbb{Z}} (-1)^m c_{1-m} \alpha_{m+2k}^{n+1}.
\end{align*}
\]

(18)

In order to perform calculations in \( L^2[0,1] \), we define the periodized wavelet \( \psi_{n,k} \) and the periodized scaling function \( \phi_{n,k} \) by equations (3) and (4), only now using the Daubechies wavelet \( \psi \) and scaling function \( \phi \) in place of the Haar wavelet and scaling function. Theorem 2 remains valid using these periodic wavelets, but the proof is more involved (see section 4.5 of [5] or section 3.11 of [8]). Therefore, for each \( f \in L^2[0,1] \) we can write

\[
f(x) = \tilde{\alpha}_0^0 + \sum_{n=0}^{2^n-1} \sum_{k=0}^{2^n-1} \tilde{\beta}_k^n \psi_{n,k}(x),
\]

(19)

where \( \tilde{\alpha}_0^0 = \int_0^1 f(x) dx \) and \( \tilde{\beta}_k^n = \int_0^1 f(x) \psi_{n,k}(x) dx \). We also define the coefficients \( \tilde{\alpha}_k^n \) by \( \tilde{\alpha}_k^n = \int_0^1 f(x) \tilde{\phi}_{n,k}(x) dx \). And, we periodically extend \( f \) with period 1, also denoting this periodic extension by \( f \). Then, for \( n \geq 0 \) and \( k = 0, 1, \ldots, 2^n - 1 \), we have

\[
\tilde{\alpha}_k^n = \int_0^1 f(x) \phi_{n,j+k}(x) \phi_{n,j}(x) dx
\]

(20)

\[
= \sum_{j \in \mathbb{Z}} \int_{j}^{j+1} f(x) \phi_{n,j+k}(x) \phi_{n,j}(x) dx
\]

\[
= \alpha_k^n.
\]

Similar arguments show that \( \tilde{\beta}_n^0 = \beta_0^0 \) and \( \tilde{\beta}_{k+2^n}^0 = \beta_k^0 \) and \( \tilde{\beta}_{k+2^n}^{n+1} = \beta_k^{n+1} \) for \( n \geq 0 \) and \( k = 0, 1, \ldots, 2^n - 1 \). After periodizing (11) and (12), it follows that

\[
\tilde{\alpha}_k^n = \sum_{m \in \mathbb{Z}} c_m \tilde{\alpha}_{m+2k}^{n+1},
\]

(21)

\[
\tilde{\beta}_k^n = \sum_{m \in \mathbb{Z}} (-1)^m c_{1-m} \tilde{\alpha}_{m+2k}^{n+1}.
\]

**Remark.** In the section "The Haar Wavelet" we saw, for the Haar wavelet \( \psi \), that \( \psi_{n,k}(x) = 2^{n/2} \psi(2^n x - k) \) for \( n \geq 0 \) and \( k = 0, 1, \ldots, 2^n - 1 \). Almost exactly the same result holds for the Daubechies wavelets. For instance, if \( \psi \) is the Daubechies wavelet, then \( \psi \) is supported on \([0, 3]\). It follows, for \( n \geq 2 \), that on the unit interval \( \psi_{n,0} \) is supported on \([0, 3 \cdot 2^{-n}]\). On the unit interval, we then have \( \psi_{n,k}(x) = 2^{n/2} \psi(2^n x - k) \) for \( k = 0, 1, \ldots, 2^n - 3 \). Hence, for \( n \geq 2 \), the periodized Daubechies wavelet functions \( \psi_{n,k} \) are identical to the Daubechies wavelets.

We can discretize the series in (19) by analogy with the Haar series. The coefficient relations in (21) yield a fast wavelet transform, \( W \), an orthogonal matrix defined by

\[
W = (W_2 \oplus IM_{M-2}) \cdot \cdots \cdot (P_{M/2} \oplus IM_{M/2})(W_{M/2} \oplus IM_{M/2}) \cdot PM \cdot WM
\]

where each matrix \( W_N \) is an \( N \times N \) orthogonal matrix (as follows from (13) and the second equation in (14)). The matrix \( W_N \) is used to produce the \( (N - 1) \)th level coefficients \( \{\tilde{\alpha}_k^{N-1}\} \) and \( \{\tilde{\beta}_k^{N-1}\} \) from the \( N \)th level scaling coefficients \( \{\tilde{\alpha}_k^N\} \) as follows:

\[
W_N \begin{bmatrix}
\tilde{\alpha}_0^N & \cdots & \tilde{\alpha}_{2^{N-1}}^N \\
\vdots & \ddots & \vdots \\
\tilde{\alpha}_{2^{N-1}}^N & \cdots & \tilde{\alpha}_{2^{N-1}}^{2N-1}
\end{bmatrix}^T
=egin{bmatrix}
\tilde{\alpha}_0^{N-1} & \tilde{\beta}_0^{N-1} & \cdots & \tilde{\alpha}_{2^{N-1}}^{N-1} & \tilde{\beta}_{2^{N-1}}^{N-1}
\end{bmatrix}^T.
\]
If we use the coefficients \( c_0, c_1, c_2, c_3 \) defined in (16), then for \( N > 2 \), \( W_N \) has the following structure:

\[
W_N = \\
\begin{pmatrix}
  c_0 & c_1 & c_2 & c_3 \\
  c_3 & c_0 & c_1 & c_2 \\
  c_2 & c_3 & c_0 & c_1 \\
  c_1 & c_2 & c_3 & c_0 \\
\end{pmatrix}
\]

For \( N = 2 \), \( W_2 = \left( \begin{array}{cccc}
  c_0^+ c_0 & c_1^+ c_1 & c_1^+ c_0 & c_0^+ c_1 \\
  c_0^+ c_0 & c_1^+ c_1 & c_1^+ c_0 & c_0^+ c_1 \\
  c_0^+ c_0 & c_1^+ c_1 & c_1^+ c_0 & c_0^+ c_1 \\
  c_0^+ c_0 & c_1^+ c_1 & c_1^+ c_0 & c_0^+ c_1 \\
\end{array} \right). \) For other Daubechies wavelets, there are other finite coefficient sequences \( \{c_k\} \), and the matrices \( W_N \) are defined similarly. The permutation matrix \( P_N \) is the same one that we defined for the Haar transform and is used to sort the \((N - 1)^{th}\) level coefficients so that \( W_{N-1} \) can be applied to the scaling coefficients \( \{\alpha_N^{(N-1)}\} \).

As initial data for the wavelet transform we can, as we did for the Haar transform, use discrete data of the form \( \{f_j\}_{j=0}^{M-1} \). The equations in (15) then provide a discrete analog of the zero moment conditions in (9) [for \( L = 2 \)]; hence the wavelet coefficients will be 0 where the data is linear. In the last section, we saw how this can produce effective compression of signals when just the 0th and 1st moments of \( \psi \) are 0.

It is often the case that the initial data are values of a measured signal, i.e. \( \{f_j\} = \{F(x_k)\} \), for \( x_k = k2^{-n} \), where \( F \) is a signal obtained from a measurement process. As shown in the previous section, we can interpret the behavior of the discrete case based on properties of the function \( F \). A measured signal \( F \) is often described by a convolution: \( F(x) = \int_{-\infty}^{\infty} g(t) \mu(x-t) \, dt \), where \( g \) is the signal being measured and \( \mu \) is called the instrument function. Such convolutions generally have greater regularity than a typical function in \( L^2(\mathbb{R}) \). For instance, if \( g \in L^1(\mathbb{R}) \) and is supported on a finite interval and \( \mu = 1_{[r,\infty)} \) for some positive \( r \), then \( F \) is continuous and supported on a finite interval. By a linear change of variables, we may then assume that \( F \) is supported on \([0,1]\). The data \( \{F(x_k)\} \) then provide approximations for the highest level scale coefficients \( \{\alpha_k^R\} \). If we assume that \( F \) has period 1, then

\[
\alpha_k^R = 2^{-R/2} \int_{-\infty}^{\infty} F(x) 2^R \phi \left( 2^R (x - x_k) \right) \, dx \\
\approx C_M F(x_k).
\]

This approximation will hold for all period 1 continuous functions \( F \) and will be more accurate the larger the value of \( 2^R = M \).

The higher the order of a Daubechies wavelet, the more of its moments are zero. A Daubechies wavelet of order 2L is defined by 2L nonzero coefficients \( \{c_k\} \), has its first \( L \) moments equal to zero, and is supported on the interval \([0, 2L - 1]\). Generally speaking, the more moments that are zero, the more wavelet coefficients that are nearly vanishing for smooth functions \( F \). This follows from considering Taylor expansions. Suppose \( F(x) \) has an \( L \)-term Taylor expansion about the point \( x_k = k2^{-n} \).

That is, \( F(x) = \sum_{j=0}^{L-1} \frac{1}{j!} F^{(j)}(x_k) (x - x_k)^j + \frac{1}{L!} F^{(L)}(t_k) (x - x_k)^L \) where \( t_k \) lies between \( x \) and \( x_k \). Suppose also that \( \psi \) is supported on \([-a,a]\) and that \( \psi \) has its first \( L \) moments vanishing. In particular, \( \psi \) is supported on \([k-a, k+a]2^{-n}\). It then follows that

\[
|\beta_k^R| \leq \frac{B}{\sqrt{L + 1/2 L!}} \left( \frac{a}{2^n} \right)^{L+1/2}.
\]

This inequality shows why \( \psi(x) \) having zero moments produces a large number of small wavelet coefficients. If \( F \) has some smoothness on an interval \((c,d)\), then wavelet coefficients \( \beta_k^R \) corresponding to the basis functions \( \tilde{\psi} (2^n (x - x_k)) \) whose supports are contained in \((c,d)\) will approach 0 rapidly as \( n \) increases to \( \infty \).

In addition to the Daubechies wavelets, there is another class of compactly supported wavelets called coiflets. These wavelets are also constructed using the method outlined above. A coiflet of order 3L is defined by 3L nonzero coefficients \( \{c_k\} \) and has its first \( L \) moments equal to zero and is supported on the interval \([-L, 2L - 1]\). A coiflet of order 3L is distinguished from a Daubechies wavelet of order 2L in that, in addition to \( \psi \) having its first \( L \) moments equal to zero, the scaling function \( \phi \) for the coiflet also has \( L - 1 \) moments vanishing. In particular, \( \int_{-\infty}^{\infty} x^j \phi(x) \, dx = 0 \), for \( j = 1, \ldots, L - 1 \). For a coiflet of order 3L, supported on \([-a,a]\), an argument similar to the one that proves (22) shows that

\[
|\alpha_k^R - C_M F(x_k)| \leq \frac{B}{\sqrt{L + 1/2 L!}} \left( \frac{a}{2^n} \right)^{L+1/2}.
\]

This inequality provides a stronger theoretical justification for using the data \( \{C_M F(x_k)\} \) in place of
the highest level scaling coefficients, beyond the argument we gave above for Daubechies wavelets. The construction of coiflets was first carried out by Daubechies and named after Coifman (who first suggested them).

**Compression of Signals**

One of the most important applications of wavelet analysis is to the compression of signals. As an example, let us use a Daub4 series to compress the signal 

\[ \{ f_j = F(j/1024) \}_{j=0}^{1023} \]

where 

\[ F(x) = -\log |x - 0.2| \]

See Figure 5(a)(top). For this signal, a partial sum containing 99% of the energy required only 37 coefficients (see Figure 5(b)(top)). It certainly provides a visually acceptable approximation of \( \{ f_j \} \). In particular, the sharp maximum in the signal near \( x = 0.2 \) seems to be reproduced quite well. The compression ratio is 1024:37 \( \approx 27:1 \), which is an excellent result considering that we also have 99% accuracy. In addition, wavelet analysis has identified the singularities of \( F \). Notice in Figure 5(b)(bottom) the peak in the wavelet coefficients is near \( x = 0.2 \), where \( F \) has a singularity, and the largest wavelet coefficient is near \( x = 1 \), where the periodic extension of \( F \) has a jump discontinuity.

Turning to Fourier series, since the even periodic extension is continuous, we used a discrete Fourier cosine series to compress this signal. In Figure 5(a)(bottom) we show a 257-term discrete Fourier cosine series partial sum for \( \{ f_j \} \). Even using seven times as many coefficients as the wavelet series, the cosine series cannot reproduce the sharp peak in the signal. Better results could be obtained in this case by either segmenting the interval and performing a cosine expansion on each segment, or by using a smoother version of the same idea involving local cosine bases [3, 9, 12, 1, 5].

One way to quantify the accuracy of these approximations is to use relative R.M.S. differences. Given two sets of data \( \{ f_j \}_{j=0}^{M-1} \) and \( \{ g_j \}_{j=0}^{M-1} \), their relative R.M.S. difference, relative to \( \{ f_j \} \), is defined by

\[
D(f, g) = \sqrt{\frac{\sum_{j=0}^{M-1} |f_j - g_j|^2}{\sum_{j=0}^{M-1} |f_j|^2}}.
\]

For the example above, if we denote the wavelet approximation by \( f^w \), then \( D(f, f^w) = 9.8 \times 10^{-3} \). For the Fourier cosine series approximation, call it \( f^c \), we have \( D(f, f^c) = 2.7 \times 10^{-2} \). A rule of thumb for a visually acceptable approximation is to have a relative R.M.S. difference of less than \( 10^{-2} \). The approximations in this example are consistent with this rule of thumb.

We can also do more localized analysis with R.M.S. differences. For example, over the subinterval \([.075, .325]\) centered on the singularity of \( F \), we find that \( D(f, f^w) = 9.7 \times 10^{-3} \) and \( D(f, f^c) = 3.2 \times 10^{-2} \). These numbers confirm our visual impression that the wavelet series does a better job reproducing the sharp peak in the signal. Or, using the subinterval \([.25, .75]\), we get \( D(f, f^w) = 1.0 \times 10^{-2} \) and \( D(f, f^c) = 3.3 \times 10^{-3} \), confirming our impression that both series do an adequate job approximating \( \{ f_j \} \) over this subinterval.
Although in the examples we have discussed so far Fourier analysis did not compress the signals very well, we do not wish to create the impression that this will always be true. In fact, if a signal is composed of relatively few sinusoids, then Fourier analysis will provide very good compression. For example, consider the signal \( \{ f_j = f(j/1024) \}_{j=0}^{1023} \) where \( f(x) \) is defined in (1) with \( \nu = 280 \). The Fourier coefficients for \( f \) are graphed in Figure 1(b)\[top\]. They tend rapidly to 0 away from the frequencies \( \pm 280 \); hence the signal is composed of relatively few sinusoids. By computing a Fourier series partial sum that uses only the 122 Fourier coefficients whose frequencies are within \( \pm 30 \) of \( \pm 280 \), we obtained a signal \( g \) that was visually indistinguishable from the original signal. In fact, 
\[ D(f,g) = 5.1 \times 10^{-3} \]
we shall refer to as a blip. Notice that each blip is concentrated around \( x = k/8 \), since 
\[ e^{-640 \pi (x-k/8)^2} \]
rapidly decreases to 0 away from \( x = k/8 \). This signal can be interpreted as representing the bit sequence 1011011. In Figure 6(a)\[bottom\] we show this signal after it has been corrupted by adding noise. In Figure 6(b)\[top\] we show the coiflet30 wavelet coefficients for the original signal. The rationale for using wavelets to remove the noise is that the original signal’s wavelet coefficients are closely correlated with the points near \( x = k/8 \) where the blips are concentrated. To demonstrate this, we show in Figure 6(b)\[middle\] a graph of the 7th level wavelet coefficients \( \{ \beta_k^7 \} \) corresponding to the positions \( \{ k2^{-7} \}_{k=0}^{2^7-1} \) on the unit interval. Comparing this to Figure 6(a)\[top\], we can see that the positions of this level’s largest magnitude wavelet coefficients are closely correlated with the positions of the blips. Similar graphs could also be drawn for other levels, but the 7th level coefficients have the largest magnitude. In Figure 6(b)\[bottom\] we show the coiflet30 transform of the noisy signal. In spite of the noise, the 7th level coefficients clearly stand out, although in a distorted form. By introducing a threshold, in this case 0.15, we can retain these 7th level coefficients and remove
most of the noise. In Figure 7(a)[top] we show the reconstructed signal obtained by computing a partial sum using only those coefficients whose magnitudes do not fall below 0.15. This reconstruction is not a flawless reproduction of the original signal, but nevertheless the amount of noise has been greatly reduced, and the bit sequence 101011 can be determined.

In Figure 7(a)[bottom] we show the denoised signal obtained by filtering the Fourier coefficients of the noisy signal (see Figure 7(b)[top]) using the method of denoising described in the section "Frequency Information, Denoising". In contrast to the wavelet denoising, the Fourier denoising has retained a significant amount of noise in the spaces between the blips. The source of this retained noise is that most of the original noise’s Fourier coefficients are of uniform magnitude distributed across all frequencies. Consequently, the filter preserves noise coefficients corresponding to frequencies that were not present in the original signal. These coefficients generate sinusoids that oscillate across the entire interval [0, 1]. The noise’s wavelet coefficients also have almost uniform magnitude, but the thresholding process eliminates all, except the ones modifying the 7th level coefficients of the original signal. Since these coefficients’ wavelet basis functions are compactly supported, this causes distortions in the recovered signal that are limited to neighborhoods of the positions of the 7th level coefficients. Consequently, there is still noise distorting the blips, but very little noise in between them.

It is also interesting to observe that the wavelet reconstructed signal and the original signal have similar frequency content. In Figure 7(b)[middle] and Figure 7(b)[bottom], we have graphed the moduli-squared of the Fourier coefficients of the original signal and of the wavelet denoised signal, respectively. These graphs show that the frequencies of the wavelet reconstruction are, like the frequencies of the original signal, concentrated around ±80 with the highest magnitude frequencies located precisely at ±80. This shows that the coiflet3 wavelet has the ability to extract frequency information. Much work has been done in refining this ability, including the development of another class of bases called wavelet packets [12, 9, 3, 5].

Conclusion
In this paper we have tried to show how the two methodologies of Fourier analysis and wavelet analysis are used for various kinds of work. Of course, we have only scratched the surface of both fields. Much more information can be found in the references and their bibliographies.

References
This spectral density is shown in Figure 10. From the shape of this density we see that in order to obtain segments in which the spectrum is flat, we need to partition the spectrum finely at low frequencies, but only coarsely at high frequencies. The subbands we obtain by this procedure will be approximately vectors of white noise with variances proportional to the power spectrum over their frequency range. We can use a procedure similar to that described for the KLT for coding the output. As we will see below, this particular partition of the spectrum is closely related to the wavelet transform.

4 Wavelets: A Different Perspective

4.1 Multiresolution Analyses

The discussion so far has been motivated by probabilistic considerations. We have been assuming our images can be reasonably well-approximated by Gaussian random vectors with a particular covariance structure. The use of the wavelet transform in image coding is motivated by a rather different perspective, that of approximation theory. We assume that our images are locally smooth functions and can be well-modeled as piecewise polynomials. Wavelets provide an efficient means for approximating such functions with a small number of basis elements. This new perspective provides some valuable insights into the coding process and has motivated some significant advances.

We motivate the use of the wavelet transform in image coding using the notion of a multiresolution analysis. Suppose we want to approximate a continuous-valued square-integrable function $f(x)$ using a discrete set of values. For example, $f(x)$ might be the brightness of a one-dimensional image. A natural set of values to use to approximate $f(x)$ is a set of regularly-spaced, weighted local averages of $f(x)$ such as might be obtained from the sensors in a digital camera.

A simple approximation of $f(x)$ based on local averages is a step function approximation. Let $\phi(x)$ be the box function given by $\phi(x) = 1$ for $x \in [0, 1)$ and 0 elsewhere. A step function approximation to $f(x)$ has the form

$$ Af(x) = \sum_n f_n \phi(x - n), \quad (1.22) $$

where $f_n$ is the height of the step in $[n, n + 1)$. A natural value for the heights $f_n$ is simply the average value of $f(x)$ in the interval $[n, n + 1)$. This gives $f_n = \int_n^{n+1} f(x) dx$.

We can generalize this approximation procedure to building blocks other than the box function. Our more generalized approximation will have the
form
\[ Af(x) = \sum_{n} \langle \hat{\phi}(x - n), f(x) \rangle \phi(x - n). \] (1.23)

Here \( \hat{\phi}(x) \) is a weight function and \( \phi(x) \) is an interpolating function chosen so that \( \langle \phi(x), \hat{\phi}(x - n) \rangle = \delta[n] \). The restriction on \( \phi(x) \) ensures that our approximation will be exact when \( f(x) \) is a linear combination of the functions \( \phi(x - n) \). The functions \( \phi(x) \) and \( \hat{\phi}(x) \) are normalized so that \( \int |\phi(x)|^2 dx = \int |\hat{\phi}(x)|^2 dx = 1 \). We will further assume that \( f(x) \) is periodic with an integer period so that we only need a finite number of coefficients to specify the approximation \( Af(x) \).

We can vary the resolution of our approximations by dilating and contracting the functions \( \phi(x) \) and \( \hat{\phi}(x) \). Let \( \phi^j(x) = 2^j \phi(2^j x) \) and \( \hat{\phi}^j(x) = 2^j \hat{\phi}(2^j x) \). We form the approximation \( A^j f(x) \) by projecting \( f(x) \) onto the span of the functions \( \{\phi^j(x - 2^{-j} k)\}_{k \in \mathbb{Z}} \), computing
\[ A^j f(x) = \sum_k \langle f(x), \hat{\phi}^j(x - 2^{-j} k) \rangle \phi^j(x - 2^{-j} k). \] (1.24)

Let \( V_j \) be the space spanned by the functions \( \{\phi^j(x - 2^{-j} k)\} \). Our resolution \( j \) approximation \( A^j f \) is simply a projection (not necessarily an orthogonal one) of \( f(x) \) onto the span of the functions \( \phi^j(x - 2^{-j} k) \).

For our box function example, the approximation \( A^j f(x) \) corresponds to an orthogonal projection of \( f(x) \) onto the space of step functions with step width \( 2^{-j} \). Figure 11 shows the difference between the coarse approximation \( A^0 f(x) \) on the left and the higher resolution approximation \( A^j f(x) \) on the right. Dilating scaling functions give us a way to construct approximations to a given function at various resolutions. An important observation is that if a given function is sufficiently smooth, the differences between approximations at successive resolutions will be small.

Constructing our function \( \phi(x) \) so that approximations at scale \( j \) are special cases of approximations at scale \( j + 1 \) will make the analysis of differences of functions at successive resolutions much easier. The function \( \phi(x) \) from our box function example has this property, since step functions with width \( 2^{-j} \) are special cases of step functions with width \( 2^{-j-1} \).

For such \( \phi(x) \)'s the spaces of approximations at successive scales will be nested, i.e. we have \( V_j \subset V_{j+1} \).

The observation that the differences \( A^{j+1} f - A^j f \) will be small for smooth functions is the motivation for the Laplacian pyramid [26], a way of transforming an image into a set of small coefficients. The 1-D analog of the procedure is as follows: we start with an initial discrete representation of a function, the \( N \) coefficients of \( A^j f \). We first split this function into the sum
\[ A^j f(x) = A^{j-1} f(x) + [A^j f(x) - A^{j-1} f(x)]. \] (1.25)

Because of the nesting property of the spaces \( V_j \), the difference \( A^j f(x) - A^{j-1} f(x) \) can be represented exactly as a sum of \( N \) translates of the func-
A continuous function \( f(x) \) (plotted as a dotted line) and box function approximations (solid lines) at two resolutions. On the left is the coarse approximation \( A^0 f(x) \) and on the right is the higher resolution approximation \( A^1 f(x) \).

The key point is that the coefficients of these \( \phi^j(x) \) translates will be small provided that \( f(x) \) is sufficiently smooth, and hence easy to code. Moreover, the dimension of the space \( V_{j-1} \) is only half that of the space \( V_j \), so we need only \( \frac{N}{2} \) coefficients to represent \( A^{j-1} f \). (In our box-function example, the function \( A^{j-1} f \) is a step function with steps twice as wide as \( A^j f \), so we need only half as many coefficients to specify \( A^{j-1} f \).) We have partitioned \( A^j f \) into \( N \) difference coefficients that are easy to code and \( \frac{N}{2} \) coarse-scale coefficients. We can repeat this process on the coarse-scale coefficients, obtaining \( \frac{N}{2} \) easy-to-code difference coefficients and \( \frac{N}{4} \) coarser scale coefficients, and so on. The end result is \( 2N - 1 \) difference coefficients and a single coarse-scale coefficient.

Burt and Adelson [26] have employed a two-dimensional version of the above procedure with some success for an image coding scheme. The main problem with this procedure is that the Laplacian pyramid representation has more coefficients to code than the original image. In 1-D we have twice as many coefficients to code, and in 2-D we have \( \frac{4}{3} \) as many.

### 4.2 Wavelets

We can improve on the Laplacian pyramid idea by finding a more efficient representation of the difference \( D^{j-1} f = A^j f - A^{j-1} f \). The idea is that to decompose a space of fine-scale approximations \( V_j \) into a direct sum of two subspaces, a space \( V_{j-1} \) of coarser-scale approximations and its complement, \( W_{j-1} \). This space \( W_{j-1} \) is a space of differences between coarse and fine-scale approximations. In particular, \( A^j f - A^{j-1} f \in W_{j-1} \) for any \( f \). Elements of the space can be thought of as the additional details that must be supplied to generate a finer-scale approximation from a coarse one.

Consider our box-function example. If we limit our attention to functions on the unit interval, then the space \( V_j \) is a space of dimension \( 2^j \). We can
decompose $V_j$ into the space $V_{j-1}$, the space of resolution $2^{-j+1}$ approximations, and $W_{j-1}$, the space of details. Because $V_{j-1}$ is of dimension $2^{j-1}$, $W_{j-1}$ must also have dimension $2^{j-1}$ for the combined space $V_j$ to have dimension $2^j$. This observation about the dimension of $W_j$ provides us with a means to circumvent the Laplacian pyramid’s problems with expansion.

Recall that in the Laplacian pyramid we represent the difference $D^{j-1}f$ as a sum of $N$ fine-scale basis functions $\phi^j(x)$. This is more information than we need, however, because the space of functions $D^{j-1}f$ is spanned by just $\frac{N}{2}$ basis functions. Let $c_k^j$ be the expansion coefficient $\langle \phi^j(x - 2^{-j}k), f(x) \rangle$ in the resolution $j$ approximation to $f(x)$. For our step functions, each coefficient $c_k^{j-1}$ is the average of the coefficients $c_{2k}^j$ and $c_{2k+1}^j$ from the resolution $j$ approximation. In order to reconstruct $A^j f$ from $A^{j+1} f$, we only need the $\frac{N}{2}$ differences $c_{2k+1}^j - c_{2k}^j$. Unlike the Laplacian pyramid, there is no expansion in the number of coefficients needed if we store these differences together with the coefficients for $A^{j-1} f$.

The differences $c_{2k+1}^j - c_{2k}^j$ in our box function example correspond (up to a normalizing constant) to coefficients of a basis expansion of the space of details $W_{j-1}$. Mallat has shown that in general the basis for $W_j$ consists of translates and dilates of a single prototype function $\psi(x)$, called a wavelet [27]. The basis for $W_j$ is of the form $\psi^j(x - 2^{-j}k)$ where $\psi^j(x) = 2^{-j/2} \psi(x)$. 

**FIGURE 12.** $D^0 f(x)$, the difference between the coarse approximation $A^0 f(x)$ and the finer scale approximation $A^1 f(x)$ from figure 11.

**FIGURE 13.** The Haar scaling function and wavelet.
Figure 13 shows the scaling function (a box function) for our box function example together with the corresponding wavelet, the Haar wavelet. Figure 12 shows the function $D^0 f(x)$, the difference between the approximations $A^1 f(x)$ and $A^1 f(x)$ from Figure 11. Note that each of the intervals separated by the dotted lines contains a translated multiple of $\psi(x)$.

The dynamic range of the differences $D^0 f(x)$ in Figure 12 is much smaller than that of $A^1 f(x)$. As a result, it is easier to code the expansion coefficients of $D^0 f(x)$ than to code those of the higher resolution approximation $A^1 f(x)$. The splitting $A^1 f(x)$ into the sum $A^0 f(x) + D^0 f(x)$ performs a packing much like that done by the Karhunen-Loève transform. For smooth functions $f(x)$ the result of the splitting of $A^1 f(x)$ into a sum of a coarser approximation and details is that most of the variation is contained in $A^0 f$, and $D^0 f$ is near zero. By repeating this splitting procedure, partitioning $A^0 f(x)$ into $A^{-1} f(x) + D^{-1} f(x)$, we obtain the wavelet transform. The result is that an initial function approximation $A^j f(x)$ is decomposed into the telescoping sum

$$A^j f(x) = D^{j-1} f(x) + D^{j-2} f(x) + \ldots + D^{j-n} f(x) + A^{j-n} f(x).$$

The coefficients of the differences $D^{j-k} f(x)$ are easier to code than the expansion coefficients of the original approximation $A^j f(x)$, and there is no expansion of coefficients as in the Laplacian pyramid.

### 4.3 Recurrence Relations

For the repeated splitting procedure above to be practical, we will need an efficient algorithm for obtaining the coefficients of the expansions $D^{j-k} f$ from the original expansion coefficients for $A^j f$. A key property of our scaling functions makes this possible.

One consequence of our partitioning of the space of resolution $j$ approximations, $V_j$, into a space of resolution $j-1$ approximations $V_{j-1}$ and resolution $j-1$ details $W_{j-1}$ is that the scaling functions $\phi(x)$ possess self-similarity properties. Because $V_{j-1} \subset V_j$, we can express the function $\phi_{j-1}(x)$ as a linear combination of the functions $\phi_j(x-n)$. In particular we have

$$\phi(x) = \sum_k h_k \phi(2x - k).$$

Similarly, we have

$$\tilde{\phi}(x) = \sum_k \tilde{h}_k \tilde{\phi}(2x - k)$$

$$\psi(x) = \sum_k g_k \phi(2x - k)$$

$$\tilde{\psi}(x) = \sum_k \tilde{g}_k \tilde{\phi}(2x - k).$$
These recurrence relations provide the link between wavelet transforms and subband transforms. Combining (4.3) and (4.3) with (4.1), we obtain a simple means for splitting the $N$ expansion coefficients for $A^j f$ into the $\frac{N}{2}$ expansion coefficients for the coarser-scale approximation $A^{j-1} f$ and the $\frac{N}{2}$ coefficients for the details $D^{j-1} f$. Both the coarser-scale approximation coefficients and the detail coefficients are obtained by convolving the coefficients of $A^j f$ with a filter and downsampling by a factor of 2. For the coarser-scale approximation, the filter is a low-pass filter with taps given by $h_{-k}$. For the details, the filter is a high-pass filter with taps $\tilde{g}_{-k}$. A related derivation shows that we can invert the split by upsampling the coarser-scale approximation coefficients and the detail coefficients by a factor of 2, convolving them with synthesis filters with taps $h_k$ and $g_k$, respectively, and adding them together.

We begin the forward transform with a signal representation in which we have very fine temporal localization of information but no frequency localization of information. Our filtering procedure splits our signal into low-pass and high-pass components and downsamples each. We obtain twice the frequency resolution at the expense of half of our temporal resolution. On each successive step we split the lowest frequency signal component into a low pass and high pass component, each time gaining better frequency resolution at the expense of temporal resolution. Figure 14 shows the partition of the time-frequency plane that results from this iterative splitting procedure. As we discussed in Section 3.5, such a decomposition, with its wide subbands in the high frequencies and narrow subbands at low frequencies leads to effective data compression for a common image model, a Gaussian random process with an exponentially decaying autocorrelation function.

The recurrence relations give rise to a fast algorithm for splitting a fine-scale function approximation into a coarser approximation and a detail function. If we start with an $N$ coefficient expansion $A^j f$, the first split requires $kN$ operations, where $k$ depends on the lengths of the filters we use. The approximation $A^{j-1}$ has $\frac{N}{2}$ coefficients, so the second split requires

![FIGURE 14. Partition of the time-frequency plane created by the wavelet transform.](image-url)
$k N^2$ operations. Each successive split requires half as much work, so the overall transform requires $O(N)$ work.

4.4 Wavelet Transforms vs. Subband Decompositions

The wavelet transform is a special case of a subband transform, as the derivation of the fast wavelet transform reveals. What, then, does the wavelet transform contribute to image coding? As we discuss below, the chief contribution of the wavelet transform is one of perspective. The mathematical machinery used to develop the wavelet transform is quite different than that used for developing subband coders. Wavelets involve the analysis of continuous functions whereas analysis of subband decompositions is more focused on discrete time signals. The theory of wavelets has a strong spatial component whereas subbands are more focused in the frequency domain.

The subband and wavelet perspectives represent two extreme points in the analysis of this iterated filtering and downsampling process. The filters used in subband decompositions are typically designed to optimize the frequency domain behavior of a single filtering and subsampling. Because wavelet transforms involve iterated filtering and downsampling, the analysis of a single iteration is not quite what we want. The wavelet basis functions can be obtained by iterating the filtering and downsampling procedure an infinite number of times. Although in applications we iterate the filtering and downsampling procedure only a small number of times, examination of the properties of the basis functions provides considerable insight into the effects of iterated filtering.

A subtle but important point is that when we use the wavelet machinery, we are implicitly assuming that the values we transform are actually fine-scale scaling function coefficients rather than samples of some function. Unlike the subband framework, the wavelet framework explicitly specifies an underlying continuous-valued function from which our initial coefficients are derived. The use of continuous-valued functions allows the use of powerful analytical tools, and it leads to a number of insights that can be used to guide the filter design process. Within the continuous-valued framework we can characterize the types of functions that can be represented exactly with a limited number of wavelet coefficients. We can also address issues such as the smoothness of the basis functions. Examination of these issues has led to important new design criteria for both wavelet filters and subband decompositions.

A second important feature of the wavelet machinery is that it involves both spatial as well as frequency considerations. The analysis of subband decompositions is typically more focused on the frequency domain. Coefficients in the wavelet transform correspond to features in the underlying function in specific, well-defined locations. As we will see below, this explicit use of spatial information has proven quite valuable in motivating
some of the most effective wavelet coders.

4.5 Wavelet Properties

There is an extensive literature on wavelets and their properties. See [28], [23], or [29] for an introduction. Properties of particular interest for image compression are the accuracy of approximation, the smoothness, and the support of these bases.

The functions \( \phi(x) \) and \( \psi(x) \) are the building blocks from which we construct our compressed images. When compressing natural images, which tend contain locally smooth regions, it is important that these building blocks be reasonably smooth. If the wavelets possess discontinuities or strong singularities, coefficient quantization errors will cause these discontinuities and singularities to appear in decoded images. Such artifacts are highly visually objectionable, particularly in smooth regions of images.

Procedures for estimating the smoothness of wavelet bases can be found in [30] and [31]. Rioul [32] has found that under certain conditions that the smoothness of scaling functions is a more important criterion than a standard frequency selectivity criterion used in subband coding.

Accuracy of approximation is a second important design criterion that has arisen from wavelet framework. A remarkable fact about wavelets is that it is possible to construct smooth, compactly supported bases that can exactly reproduce any polynomial up to a given degree. If a continuous-valued function \( f(x) \) is locally equal to a polynomial, we can reproduce that portion of \( f(x) \) exactly with just a few wavelet coefficients. The degree of the polynomials that can be reproduced exactly is determined by the number of vanishing moments of the dual wavelet \( \tilde{\psi}(x) \). The dual wavelet \( \tilde{\psi}(x) \) has \( N \) vanishing moments provided that \( \int x^k \tilde{\psi}(x) \, dx = 0 \) for \( k = 0, \ldots, N \). Compactly supported bases for \( L^2 \) for which \( \tilde{\psi}(x) \) has \( N \) vanishing moments can locally reproduce polynomials of degree \( N - 1 \).

The number of vanishing moments also determines the rate of convergence of the approximations \( A^j f \) to the original function \( f \) as the resolution goes to infinity. It has been shown that \( \| f - A^j f \| \leq C 2^{-j N} \| f^{(N)} \| \) where \( N \) is the number of vanishing moments of \( \psi(x) \) and \( f^{(N)} \) is the \( N \)th derivative of \( f \) [33, 34, 35].

The size of the support of the wavelet basis is another important design criterion. Suppose that the function \( f(x) \) we are transforming is equal to polynomial of degree \( N - 1 \) in some region. If \( \psi \) has has \( N \) vanishing moments, then any basis function for which the corresponding dual function lies entirely in the region in which \( f \) is polynomial will have a zero coefficient. The smaller the support of \( \tilde{\psi} \) is, the more zero coefficients we will obtain. More importantly, edges produce large wavelet coefficients. The larger \( \tilde{\psi} \) is, the more likely it is to overlap an edge. Hence it is important that our wavelets have reasonably small support.
Wavelet-based Image Coding: An Overview

There is a tradeoff between wavelet support and the regularity and accuracy of approximation. Wavelets with short support have strong constraints on their regularity and accuracy of approximation, but as the support is increased they can be made to have arbitrary degrees of smoothness and numbers of vanishing moments. This limitation on support is equivalent to keeping the analysis filters short. Limiting filter length is also an important consideration in the subband coding literature, because long filters lead to ringing artifacts around edges.

5 A Basic Wavelet Image Coder

State-of-the-art wavelet coders are all derived from the transform coder paradigm. There are three basic components that underly current wavelet coders: a decorrelating transform, a quantization procedure, and an entropy coding procedure. Considerable current research is being performed on all three of these components. Before we discuss state-of-the-art coders in the next sections, we will describe a basic wavelet transform coder and discuss optimized versions of each of the components.

5.1 Choice of Wavelet Basis

Deciding on the optimal wavelet basis to use for image coding is a difficult problem. A number of design criteria, including smoothness, accuracy of approximation, size of support, and filter frequency selectivity are known to be important. However, the best combination of these features is not known.

The simplest form of wavelet basis for images is a separable basis formed from translations and dilations of products of one-dimensional wavelets. Using separable transforms reduces the problem of designing efficient wavelets to a one-dimensional problem, and almost all current coders employ separable transforms. Recent work of Sweldens and Kovačević [36] simplifies considerably the design of non-separable bases, and such bases may prove more efficient than separable transforms.

The prototype basis functions for separable transforms are $\phi(x)\phi(y)$, $\phi(x)\psi(y)$, $\psi(x)\phi(y)$, and $\psi(x)\psi(y)$. Each step of the transform for such bases involves two frequency splits instead of one. Suppose we have an $N \times N$ image. First each of the $N$ rows in the image is split into a low-pass half and a high-pass half. The result is an $N \times \frac{N}{2}$ sub-image and an $N \times \frac{N}{2}$ high-pass sub-image. Next each column of the sub-images is split into a low-pass and a high-pass half. The result is a four-way partition

\[ \text{C++ source code for a coder that implements these components is available from the web site } \text{http://www.cs.dartmouth.edu/~gdavis/wavelet/wavelet.html}. \]
of the image into horizontal low-pass/vertical low-pass, horizontal high-
pass/vertical low-pass, horizontal low-pass/vertical high-pass, and horizon-
tal high-pass/vertical high-pass sub-images. The low-pass/low-pass sub-
image is subdivided in the same manner in the next step as is illustrated
in Figure 17.

Unser [35] shows that spline wavelets are attractive for coding appli-
cations based on approximation theoretic considerations. Experiments by
Riou [32] for orthogonal bases indicate that smoothness is an important
consideration for compression. Experiments by Antonini et al [37] find that
both vanishing moments and smoothness are important, and for the filters
tested they found that smoothness appeared to be slightly more impor-
tant than the number of vanishing moments. Nonetheless, Vetterli and
Herley [38] state that “the importance of regularity for signal processing
applications is still an open question.” The bases most commonly used
in practice have between one and two continuous derivatives. Additional
smoothness does not appear to yield significant improvements in coding
results.

Villasenor et al [39] have systematically examined all minimum order
biorthogonal filter banks with lengths \( \leq 36 \). In addition to the criteria
already mentioned, [39] also examines measures of oscillatory behavior and
of the sensitivity of the coarse-scale approximations \( A^j f(x) \) to translations
of the function \( f(x) \). The best filter found in these experiments was a 7/9-
tap spline variant with less dissimilar lengths from [37], and this filter is
one of the most commonly used in wavelet coders.

There is one caveat with regard to the results of the filter evaluation
in [39]. Villasenor et al compare peak signal to noise ratios generated by a
simple transform coding scheme. The bit allocation scheme they use works
well for orthogonal bases, but it can be improved upon considerably in
the biorthogonal case. This inefficient bit allocation causes some promising
biorthogonal filter sets to be overlooked.

For biorthogonal transforms, the squared error in the transform domain
is not the same as the squared error in the original image. As a result,
the problem of minimizing image error is considerably more difficult than
in the orthogonal case. We can reduce image-domain errors by performing
bit allocation using a weighted transform-domain error measure that we
discuss in section 5.5. A number of other filters yield performance com-
parable to that of the 7/9 filter of [37] provided that we do bit allocation
with a weighted error measure. One such basis is the Deslauriers-Dubuc
interpolating wavelet of order 4 [40, 41], which has the advantage of having
filter taps that are dyadic rationals. Both the spline wavelet of [37] and the
order 4 Deslauriers-Dubuc wavelet have 4 vanishing moments in both \( \psi(x) \)
and \( \tilde{\psi}(x) \), and the basis functions have just under 2 continuous derivatives
in the \( L^2 \) sense.

One new very promising set of filters has been developed by Balasingham
and Ramstad [42]. Their design procedure combines classical filter design
techniques with ideas from wavelet constructions and yields filters that perform significantly better than the popular 7/9 filter set from [37].

5.2 Boundaries
Careful handling of image boundaries when performing the wavelet transform is essential for effective compression algorithms. Naive techniques for artificially extending images beyond given boundaries such as periodization or zero-padding lead to significant coding inefficiencies. For symmetrical wavelets an effective strategy for handling boundaries is to extend the image via reflection. Such an extension preserves continuity at the boundaries and usually leads to much smaller wavelet coefficients than if discontinuities were present at the boundaries. Brislawn [43] describes in detail procedures for non-expansive symmetric extensions of boundaries. An alternative approach is to modify the filter near the boundary. Boundary filters [44, 45] can be constructed that preserve filter orthogonality at boundaries. The lifting scheme [46] provides a related method for handling filtering near the boundaries.

5.3 Quantization
Most current wavelet coders employ scalar quantization for coding. There are two basic strategies for performing the scalar quantization stage. If we knew the distribution of coefficients for each subband in advance, the optimal strategy would be to use entropy-constrained Lloyd-Max quantizers for each subband. In general we do not have such knowledge, but we can provide a parametric description of coefficient distributions by sending side information. Coefficients in the high pass subbands of a wavelet transform are known a priori to be distributed as generalized Gaussians [27] centered around zero.

A much simpler quantizer that is commonly employed in practice is a uniform quantizer with a dead zone. The quantization bins, as shown in Figure 15, are of the form \([n\Delta, (n + 1)\Delta]\) for \(n \in \mathbb{Z}\) except for the central bin \([-\Delta, \Delta]\). Each bin is decoded to the value at its center in the simplest case, or to the centroid of the bin. In the case of asymptotically high rates, uniform quantization is optimal [47]. Although in practical regimes these dead-zone quantizers are suboptimal, they work almost as well as Lloyd-

\[
\text{Dead zone}
\]

\[x=0\]

FIGURE 15. Dead-zone quantizer, with larger encoder partition around \(x = 0\) (dead zone) and uniform quantization elsewhere.
Max coders when we decode to the bin centroids \cite{48}. Moreover, dead-zone quantizers have the advantage that of being very low complexity and robust to changes in the distribution of coefficients in source. An additional advantage of these dead-zone quantizers is that they can be nested to produce an embedded bitstream following a procedure in \cite{49}.

5.4 Entropy Coding

Arithmetic coding provides a near-optimal entropy coding for the quantized coefficient values. The coder requires an estimate of the distribution of quantized coefficients. This estimate can be approximately specified by providing parameters for a generalized Gaussian or a Laplacian density. Alternatively the probabilities can be estimated online. Online adaptive estimation has the advantage of allowing coders to exploit local changes in image statistics. Efficient adaptive estimation procedures are discussed in \cite{50} and \cite{51}.

Because images are not jointly Gaussian random processes, the transform coefficients, although decorrelated, still contain considerable structure. The entropy coder can take advantage of some of this structure by conditioning the encodings on previously encoded values. A coder of \cite{49} obtains modest performance improvements using such a technique.

5.5 Bit Allocation

The final question we need to address is that of how finely to quantize each subband. As we discussed in Section 3.2, the general idea is to determine the number of bits $b_j$ to devote to coding subband $j$ so that the total distortion $\sum_j D_j(b_j)$ is minimized subject to the constraint that $\sum_j b_j \leq b$. Here $D_j(b)$ is the amount of distortion incurred in coding subband $j$ with $b$ bits. When the functions $D_j(b)$ are known in closed form we can solve the problem using the Kuhn-Tucker conditions. One common practice is to approximate the functions $D_j(b)$ with the rate-distortion function for a Gaussian random variable. However, this approximation is not very accurate at low bit rates. Better results may be obtained by measuring $D_j(b)$ for a range of values of $b$ and then solving the constrained minimization problem using integer programming techniques. An algorithm of Shoham and Gersho \cite{52} solves precisely this problem.

For biorthogonal wavelets we have the additional problem that squared error in the transform domain is not equal to squared error in the inverted image. Moulin \cite{53} has formulated a multiscale relaxation algorithm which provides an approximate solution to the allocation problem for this case. Moulin’s algorithm yields substantially better results than the naive approach of minimizing squared error in the transform domain.

A simpler approach is to approximate the squared error in the image by weighting the squared errors in each subband. The weight $w_j$ for subband
The weight \( w_j \) is equal to the sum of the squares of the values in the resulting inverse transform. We allocate bits by minimizing the weighted sum \( \sum_j w_j D_j(b_j) \) rather than the sum \( \sum_j D_j(b_j) \). Further details may be found in Naveen and Woods [54]. This weighting procedure results in substantial coding improvements when using wavelets that are not very close to being orthogonal, such as the Deslauriers-Dubuc wavelets popularized by the lifting scheme [46]. The 7/9 tap filter set of [37], on the other hand, has weights that are all nearly 1, so this weighting provides little benefit.

### 5.6 Perceptually Weighted Error Measures

Our goal in lossy image coding is to minimize visual discrepancies between the original and compressed images. Measuring visual discrepancy is a difficult task. There has been a great deal of research on this problem, but because of the great complexity of the human visual system, no simple, accurate, and mathematically tractable measure has been found.

Our discussion up to this point has focused on minimizing squared error distortion in compressed images primarily because this error metric is mathematically convenient. The measure suffers from a number of deficits, however. For example, consider two images that are the same everywhere except in a small region. Even if the difference in this small region is large and highly visible, the mean squared error for the whole image will be small because the discrepancy is confined to a small region. Similarly, errors that are localized in straight lines, such as the blocking artifacts produced by the discrete cosine transform, are much more visually objectionable than squared error considerations alone indicate.

There is evidence that the human visual system makes use of a multiresolution image representation; see [55] for an overview. The eye is much more sensitive to errors in low frequencies than in high. As a result, we can improve the correspondence between our squared error metric and perceived error by weighting the errors in different subbands according to the eye’s contrast sensitivity in a corresponding frequency range. Weights for the commonly used 7/9-tap filter set of [37] have been computed by Watson et al in [56].

### 6 Extending the Transform Coder Paradigm

The basic wavelet coder discussed in Section 5 is based on the basic transform coding paradigm, namely decorrelation and compaction of energy into a few coefficients. The mathematical framework used in deriving the wavelet transform motivates compression algorithms that go beyond the traditional
mechanisms used in transform coding. These important extensions are at the heart of modern wavelet coding algorithms of Sections 7 and 9. We take a moment here to discuss these extensions.

Conventional transform coding relies on energy compaction in an ordered set of transform coefficients, and quantizes those coefficients with a priority according to their order. This paradigm, while quite powerful, is based on several assumptions about images that are not always completely accurate. In particular, the Gaussian assumption breaks down for the joint distributions across image discontinuities. Mallat and Falzon [57] give the following example of how the Gaussian, high-rate analysis breaks down at low rates for non-Gaussian processes.

Let \( Y[n] \) be a random \( N \)-vector defined by

\[
Y[n] = \begin{cases} 
X & \text{if } n = P \\
X & \text{if } n = P + 1 \mod N \\
0 & \text{otherwise}
\end{cases}
\] (1.29)

Here \( P \) is a random integer uniformly distributed between 0 and \( N - 1 \) and \( X \) is a random variable that equals 1 or -1 each with probability \( \frac{1}{2} \). \( X \) and \( P \) are independent. The vector \( Y \) has zero mean and a covariance matrix with entries

\[
E\{Y[n]Y[m]\} = \begin{cases} 
\frac{2}{N} & \text{for } n = m \\
\frac{1}{N} & \text{for } |n - m| \in \{1, N - 1\} \\
0 & \text{otherwise}
\end{cases}
\] (1.30)

The covariance matrix is circulant, so the KLT for this process is the simply the Fourier transform. The Fourier transform of \( Y \) is a very inefficient representation for coding \( Y \). The energy at frequency \( k \) will be \( |1 + e^{-2\pi i \frac{k}{N}}|^2 \) which means that the energy of \( Y \) is spread out over the entire low-frequency half of the Fourier basis with some spill-over into the high-frequency half. The KLT has “packed” the energy of the two non-zero coefficients of \( Y \) into roughly \( \frac{2}{N} \) coefficients. It is obvious that \( Y \) was much more compact in its original form, and could be coded better without transformation: Only two coefficients in \( Y \) are non-zero, and we need only specify the values of these coefficients and their positions.

As suggested by the example above, the essence of the extensions to traditional transform coding is the idea of selection operators. Instead of quantizing the transform coefficients in a pre-determined order of priority, the wavelet framework lends itself to improvements, through judicious choice of which elements to code. This is made possible primarily because wavelet basis elements are spatially as well as spectrally compact. In parts of the image where the energy is spatially but not spectrally compact (like the example above) one can use selection operators to choose subsets of the wavelet coefficients that represent that signal efficiently. A most notable example is the Zerotree coder and its variants (Section 7).
More formally, the extension consists of dropping the constraint of linear image approximations, as the selection operator is nonlinear. The work of DeVore et al. [58] and of Mallat and Falzon [57] suggests that at low rates, the problem of image coding can be more effectively addressed as a problem in obtaining a non-linear image approximation. This idea leads to some important differences in coder implementation compared to the linear framework. For linear approximations, Theorems 3.1 and 3.3 in Section 3.1 suggest that at low rates we should approximate our images using a fixed subset of the Karhunen-Loève basis vectors. We set a fixed set of transform coefficients to zero, namely the coefficients corresponding to the smallest eigenvalues of the covariance matrix. The non-linear approximation idea, on the other hand, is to approximate images using a subset of basis functions that are selected adaptively based on the given image. Information describing the particular set of basis functions used for the approximation, called a significance map, is sent as side information. In Section 7 we describe zerotrees, a very important data structure used to efficiently encode significance maps.

Our example suggests that a second important assumption to relax is that our images come from a single jointly Gaussian source. We can obtain better energy packing by optimizing our transform to the particular image at hand rather than to the global ensemble of images. The KLT provides efficient variance packing for vectors drawn from a single Gaussian source. However, if we have a mixture of sources the KLT is considerably less efficient. Frequency-adaptive and space/frequency-adaptive coders decompose images over a large library of different bases and choose an energy-packing transform that is adapted to the image itself. We describe these adaptive coders in Section 8.

Trellis coded quantization represents a more drastic departure from the transform coder framework. While TCQ coders operate in the transform domain, they effectively do not use scalar quantization. Trellis coded quantization captures not only correlation gain and fractional bitrates, but also the packing gain of VQ. In both performance and complexity, TCQ is essentially VQ in disguise.

The selection operator that characterizes the extension to the transform coder paradigm generates information that needs to be conveyed to the decoder as “side information”. This side information can be in the form of zerotrees, or more generally energy classes. Backward mixture estimation represents a different approach: it assumes that the side information is largely redundant and can be estimated from the causal data. By cutting down on the transmitted side information, these algorithms achieve a remarkable degree of performance and efficiency.

For reference, Table 1.1 provides a comparison of the peak signal to
TABLE 1.1. Peak signal to noise ratios in decibels for coders discussed in the paper. Higher values indicate better performance.

<table>
<thead>
<tr>
<th>Type of Coder</th>
<th>Lena (bits/pixel)</th>
<th>Barbara (bits/pixel)</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPEG [59]</td>
<td>37.9</td>
<td>33.1</td>
</tr>
<tr>
<td>Optimized JPEG [60]</td>
<td>39.6</td>
<td>35.9</td>
</tr>
<tr>
<td>Baseline Wavelet [61]</td>
<td>39.4</td>
<td>35.1</td>
</tr>
<tr>
<td>Zerotree (Shapiro) [62]</td>
<td>39.6</td>
<td>35.1</td>
</tr>
<tr>
<td>Zerotree (Said &amp; Pearlman) [63]</td>
<td>40.5</td>
<td>36.9</td>
</tr>
<tr>
<td>Zerotree (R/D optimized) [64]</td>
<td>40.5</td>
<td>37.0</td>
</tr>
<tr>
<td>Frequency-adaptive [65]</td>
<td>39.3</td>
<td>36.4</td>
</tr>
<tr>
<td>Space-frequency adaptive [66]</td>
<td>40.1</td>
<td>37.0</td>
</tr>
<tr>
<td>Frequency-adaptive + zerotrees [67]</td>
<td>40.6</td>
<td>37.7</td>
</tr>
<tr>
<td>TCQ subband [68]</td>
<td>41.1</td>
<td>–</td>
</tr>
<tr>
<td>Bkwd. mixture estimation (EQ) [69]</td>
<td>40.9</td>
<td>–</td>
</tr>
</tbody>
</table>

noise ratios for the coders we discuss in the paper. The test images are the $512 \times 512$ Lena image and the $512 \times 512$ Barbara image. Figure 16 shows the Barbara image as compressed by JPEG, a baseline wavelet transform coder, and the zerotree coder of Said and Pearlman [63]. The Barbara image is particularly difficult to code, and we have compressed the image at a low rate to emphasize coder errors. The blocking artifacts produced by the discrete cosine transform are highly visible in the image on the top right. The difference between the two wavelet coded images is more subtle but quite visible at close range. Because of the more efficient coefficient encoding (to be discussed below), the zerotree-coded image has much sharper edges and better preserves the striped texture than does the baseline transform coder.

7 Zerotree Coding

The rate-distortion analysis of the previous sections showed that optimal bitrate allocation is achieved when the signal is divided into subbands such that each subband contains a “white” signal. It was also shown that for typical signals of interest, this leads to narrower bands in the low frequencies and wider bands in the high frequencies. Hence, wavelet transforms have very good energy compaction properties.

This energy compaction leads to efficient utilization of scalar quantizers. However, a cursory examination of the transform in Figure 17 shows that a significant amount of structure is present, particularly in the fine scale coefficients. Wherever there is structure, there is room for compression, and

---

8 More current numbers may be found on the web at http://www.icsl.ucla.edu/~ipl/psnr_results.html
FIGURE 16. Results of different compression schemes for the $512 \times 512$ Barbara test image at 0.25 bits per pixel. Top left: original image. Top right: baseline JPEG, PSNR = 24.4 dB. Bottom left: baseline wavelet transform coder [61], PSNR = 26.6 dB. Bottom right: Said and Pearlman zerotree coder, PSNR = 27.6 dB.
advanced wavelet compression algorithms all address this structure in the higher frequency subbands.

One of the most prevalent approaches to this problem is based on exploiting the relationships of the wavelet coefficients across bands. A direct visual inspection indicates that large areas in the high frequency bands have little or no energy, and the small areas that have significant energy are similar in shape and location, across different bands. These high-energy areas stem from poor energy compaction close to the edges of the original image. Flat and slowly varying regions in the original image are well-described by the low-frequency basis elements of the wavelet transform (hence leading to high energy compaction). At the edge locations, however, low-frequency basis elements cannot describe the signal adequately, and some of the energy leaks into high-frequency coefficients. This happens similarly at all scales, thus the high-energy high-frequency coefficients representing the edges in the image have the same shape.

Our \textit{a priori} knowledge that images of interest are formed mainly from flat areas, textures, and edges, allows us to take advantage of the resulting cross-band structure. Zerotree coders combine the idea of cross-band correlation with the notion of coding zeros jointly (which we saw previously in the case of JPEG), to generate very powerful compression algorithms.

The first instance of the implementation of zerotrees is due to Lewis and Knowles [70]. In their algorithm the image is represented by a tree-structured data construct (Figure 18). This data structure is implied by a dyadic discrete wavelet transform (Figure 19) in two dimensions. The root node of the tree represents the scaling function coefficient in the lowest frequency band, which is the parent of three nodes. Nodes inside the tree correspond to wavelet coefficients at a scale determined by their height in the tree. Each of these coefficients has four children, which correspond to the wavelets at the next finer scale having the same location in space. These four coefficients represent the four phases of the higher resolution basis elements at that location. At the bottom of the data structure lie the leaf nodes, which have no children.

Note that there exist three such quadtrees for each coefficient in the low frequency band. Each of these three trees corresponds to one of three filtering orderings: there is one tree consisting entirely of coefficients arising from horizontal high-pass, vertical low-pass operation (HL); one for horizontal low-pass, vertical high-pass (LH), and one for high-pass in both directions (HH).

The zerotree quantization model used by Lewis and Knowles was arrived at by observing that often when a wavelet coefficient is small, its children on the wavelet tree are also small. This phenomenon happens because significant coefficients arise from edges and texture, which are local. It is not difficult to see that this is a form of conditioning. Lewis and Knowles took this conditioning to the limit, and assumed that insignificant parent nodes always imply insignificant child nodes. A tree or subtree that contains (or
1. Wavelet-based Image Coding: An Overview

FIGURE 17. Wavelet transform of the image “Lena.”

FIGURE 18. Space-frequency structure of wavelet transform
is assumed to contain) only insignificant coefficients is known as a zerotree.

Lewis and Knowles used the following algorithm for the quantization of wavelet coefficients: Quantize each node according to an optimal scalar quantizer for the Laplacian density. If the node value is insignificant according to a pre-specified threshold, ignore all its children. These ignored coefficients will be decoded as zeros at the decoder. Otherwise, go to each of its four children and repeat the process. If the node was a leaf node and did not have a child, go to the next root node and repeat the process.

Aside from the nice energy compaction properties of the wavelet transform, the Lewis and Knowles coder achieves its compression ratios by joint coding of zeros. For efficient run-length coding, one needs to first find a conducive data structure, e.g. the zig-zag scan in JPEG. Perhaps the most significant contribution of this work was to realize that wavelet domain data provide an excellent context for run-length coding: not only are large run lengths of zeros generated, but also there is no need to transmit the length of zero runs, because they are assumed to automatically terminate at the leaf nodes of the tree. Much the same as in JPEG, this is a form of joint vector/scalar quantization. Each individual (significant) coefficient is quantized separately, but the symbols corresponding to small coefficients in fact are representing a vector consisting of that element and the zero run that follows it to the bottom of the tree.

While this compression algorithm generates subjectively acceptable images, its rate-distortion performance falls short of baseline JPEG, which at the time was often used for comparison purposes. The lack of sophistication in the entropy coding of quantized coefficients somewhat disadvantages this coder, but the main reason for its mediocre performance is the way it generates and recognizes zerotrees. As we have noted, whenever a coefficient is small, it is likely that its descendents are also insignificant. However, the Lewis and Knowles algorithm assumes that small parents always have small descendents, and therefore suffers large distortions when this does not hold because it zeros out large coefficients. The advantage of this method is that the detection of zerotrees is automatic: zerotrees are determined by measuring the magnitude of known coefficients. No side information is required to specify the locations of zerotrees, but this simplicity is obtained at the cost of reduced performance. More detailed analysis of this tradeoff gave
rise to the next generation of zerotree coders.

7.1 The Shapiro and Said-Pearlman Coders

The Lewis and Knowles algorithm, while capturing the basic ideas inherent in many of the later coders, was incomplete. It had all the intuition that lies at the heart of more advanced zerotree coders, but did not efficiently specify significance maps, which is crucial to the performance of wavelet coders.

A significance map is a binary function whose value determines whether each coefficient is significant or not. If not significant, a coefficient is assumed to quantize to zero. Hence a decoder that knows the significance map needs no further information about that coefficient. Otherwise, the coefficient is quantized to a non-zero value. The method of Lewis and Knowles does not generate a significance map from the actual data, but uses one implicitly, based on \textit{a priori} assumptions on the structure of the data. On the infrequent occasions when this assumption does not hold, a high price is paid in terms of distortion. The methods to be discussed below make use of the fact that, by using a small number of bits to correct mistakes in our assumptions about the occurrences of zerotrees, we can reduce the coded image distortion considerably.

The first algorithm of this family is due to Shapiro [71] and is known as the embedded zerotree wavelet (EZW) algorithm. Shapiro’s coder was based on transmitting both the non-zero data and a significance map. The bits needed to specify a significance map can easily dominate the coder output, especially at lower bitrates. However, there is a great deal of redundancy in a general significance map for visual data, and the bitrates for its representation can be kept in check by conditioning the map values at each node of the tree on the corresponding value at the parent node. Whenever an insignificant parent node is observed, it is highly likely that the descendents are also insignificant. Therefore, most of the time, a “zerotree” significance map symbol is generated. But because $p$, the probability of this event, is close to 1, its information content, $-p \log p$, is very small. So most of the time, a very small amount of information is transmitted, and this keeps the average bitrate needed for the significance map relatively small.

Once in a while, one or more of the children of an insignificant node will be significant. In that case, a symbol for “isolated zero” is transmitted. The likelihood of this event is lower, and thus the bitrate for conveying this information is higher. But it is essential to pay this price to avoid losing significant information down the tree and therefore generating large distortions.

In summary, the Shapiro algorithm uses three symbols for significance maps: zerotree, isolated zero, or significant value. But using this structure, and by conditionally entropy coding these symbols, the coder achieves very
good rate-distortion performance.

In addition, Shapiro’s coder also generates an embedded code. Coders that generate embedded codes are said to have the progressive transmission or successive refinement property. Successive refinement consists of first approximating the image with a few bits of data, and then improving the approximation as more and more information is supplied. An embedded code has the property that for two given rates $R_1 > R_2$, the rate-$R_2$ code is a prefix to the rate-$R_1$ code. Such codes are of great practical interest for the following reasons:

- The encoder can easily achieve a precise bitrate by continuing to output bits when it reaches the desired rate.
- The decoder can cease decoding at any given point, generating an image that is the best representation possible with the decoded number of bits. This is of practical interest for broadcast applications where multiple decoders with varying computational, display, and bandwidth capabilities attempt to receive the same bitstream. With an embedded code, each receiver can decode the passing bitstream according to its particular needs and capabilities.
- Embedded codes are also very useful for indexing and browsing, where only a rough approximation is sufficient for deciding whether the image needs to be decoded or received in full. The process of screening images can be speeded up considerably by using embedded codes: after decoding only a small portion of the code, one knows if the target image is present. If not, decoding is aborted and the next image is requested, making it possible to screen a large number of images quickly. Once the desired image is located, the complete image is decoded.

Shapiro’s method generates an embedded code by using a bit-slice approach (see Figure 20). First, the wavelet coefficients of the image are
indexed into a one-dimensional array, according to their order of importance. This order places lower frequency bands before higher frequency bands since they have more energy, and coefficients within each band appear in a raster scan order. The bit-slice code is generated by scanning this one-dimensional array, comparing each coefficient with a threshold $T$. This initial scan provides the decoder with sufficient information to recover the most significant bit slice. In the next pass, our information about each coefficient is refined to a resolution of $T/2$, and the pass generates another bit slice of information. This process is repeated until there are no more slices to code.

Figure 20 shows that the upper bit slices contain a great many zeros because there are many coefficients below the threshold. The role of zerotree coding is to avoid transmitting all these zeros. Once a zerotree symbol is transmitted, we know that all the descendent coefficients are zero, so no information is transmitted for them. In effect, zerotrees are a clever form of run-length coding, where the coefficients are ordered in a way to generate longer run lengths (more efficient) as well as making the runs self-terminating, so the length of the runs need not be transmitted.

The zerotree symbols (with high probability and small code length) can be transmitted again and again for a given coefficient, until it rises above the sinking threshold, at which point it will be tagged as a significant coefficient. After this point, no more zerotree information will be transmitted for this coefficient.

To achieve embeddedness, Shapiro uses a clever method of encoding the sign of the wavelet coefficients with the significance information. There are also further details of the priority of wavelet coefficients, the bit-slice coding, and adaptive arithmetic coding of quantized values (entropy coding), which we will not pursue further in this review. The interested reader is referred to [71] for more details.

Said and Pearlman [72] have produced an enhanced implementation of the zerotree algorithm, known as Set Partitioning in Hierarchical Trees (SPHIT). Their method is based on the same premises as the Shapiro algorithm, but with more attention to detail. The public domain version of this coder is very fast, and improves the performance of EZW by 0.3-0.6 dB. This gain is mostly due to the fact that the original zerotree algorithms allow special symbols only for single zerotrees, while in reality, there are other sets of zeros that appear with sufficient frequency to warrant special symbols of their own. In particular, the Said-Pearlman coder provides symbols for combinations of parallel zerotrees.

Davis and Chawla [73] have shown that both the Shapiro and the Said and Pearlman coders are members of a large family of tree-structured significance mapping schemes. They provide a theoretical framework that explains in more detail the performance of these coders and describe an algorithm for selecting a member of this family of significance maps that is optimized for a given image or class of images.
7.2 Zerotrees and Rate-Distortion Optimization

In the previous coders, zerotrees were used only when they were detected in the actual data. But consider for the moment the following hypothetical example: assume that in an image, there is a wide area of little activity, so that in the corresponding location of the wavelet coefficients there exists a large group of insignificant values. Ordinarily, this would warrant the use of a big zerotree and a low expenditure of bitrate over that area. Suppose, however, that there is a one-pixel discontinuity in the middle of the area, such that at the bottom of the would-be zerotree, there is one significant coefficient. The algorithms described so far would prohibit the use of a zerotree for the entire area.

Inaccurate representation of a single pixel will change the average distortion in the image only by a small amount. In our example we can gain significant coding efficiency by ignoring the single significant pixel so that we can use a large zerotree. We need a way to determine the circumstances under which we should ignore significant coefficients in this manner.

The specification of a zerotree for a group of wavelet coefficient is a form of quantization. Generally, the values of the pixels we code with zerotrees are non-zero, but in using a zerotrewe specify that they be decoded as zeros. Non-zerotree wavelet coefficients (significant values) are also quantized, using scalar quantizers. If we saves bitrate by specifying larger zerotrees, as in the hypothetical example above, the rate that was saved can be assigned to the scalar quantizers of the remaining coefficients, thus quantizing them more accurately. Therefore, we have a choice in allocating the bitrate among two types of quantization. The question is, if we are given a unit of rate to use in coding, where should it be invested so that the corresponding reduction in distortion is maximized?

This question, in the context of zerotree wavelet coding, was addressed by Xiong et al. [74], using well-known bit allocation techniques [1]. The central result for optimal bit allocation states that, in the optimal state, the slope of the operational rate-distortion curves of all quantizers are equal. This result is intuitive and easy to understand. The slope of the operational rate-distortion function for each quantizer tells us how many units of distortion we add/eliminate for each unit of rate we eliminate/add. If one of the quantizers has a smaller R-D slope, meaning that it is giving us less distortion reduction for our bits spent, we can take bits away from this quantizer (i.e. we can reduce its step size) and give them to the other, more efficient quantizers. We continue to do so until all quantizers have an equal slope.

Obviously, specification of zerotrees affects the quantization levels of non-zero coefficients because total available rate is limited. Conversely, specifying quantization levels will affect the choice of zerotrees because it affects the incremental distortion between zerotree quantization and scalar quantization. Therefore, an iterative algorithm is needed for rate-distortion opti-
mization. In phase one, the uniform scalar quantizers are fixed, and optimal zerotrees are chosen. In phase two, zerotrees are fixed and the quantization level of uniform scalar quantizers is optimized. This algorithm is guaranteed to converge to a local optimum [74].

There are further details of this algorithm involving prediction and description of zerotrees, which we leave out of the current discussion. The advantage of this method is mainly in performance, compared to both EZW and SPHIT (the latter only slightly). The main disadvantages of this method are its complexity, and perhaps more importantly, that it does not generate an embedded bitstream.

8 Frequency, space-frequency adaptive coders

8.1 Wavelet Packets

The wavelet transform does a good job of decorrelating image pixels in practice, especially when images have power spectra that decay approximately uniformly and exponentially. However, for images with non-exponential rates of spectral decay and for images which have concentrated peaks in the spectra away from DC, we can do considerably better.

Our analysis of Section 3.5 suggests that the optimal subband decomposition for an image is one for which the spectrum in each subband is approximately flat. The octave-band decomposition produced by the wavelet transform produces nearly flat spectra for exponentially decaying spectra. The Barbara test image shown in Figure 16 contains a narrow-band component at high frequencies that comes from the tablecloth and the striped clothing. Fingerprint images contain similar narrow-band high frequency components.

The best basis algorithm, developed by Coifman and Wickerhauser [75], provides an efficient way to find a fast, wavelet-like transform that provides a good approximation to the Karhunen-Loève transform for a given image. As with the wavelet transform, we start by assuming that a given signal corresponds to a sum of fine-scale scaling functions. The transform performs a change of basis, but the new basis functions are not wavelets but rather wavelet packets [76].

Like wavelets, wavelet packets are formed from translated and dilated linear combinations of scaling functions. However, the recurrence relations they satisfy are different, and the functions form an overcomplete set. Consider a signal of length $2^N$. The wavelet basis for such a signal consists of a scaling function and $2^N - 1$ translates and dilates of the wavelet $\psi(x)$. Wavelet packets are formed from translates and dilates of $2^N$ different prototype functions, and there are $N2^N$ different possible functions that can be used to form a basis.

Wavelet packets are formed from recurrence relations similar to those for
wavelets and generalize the theoretical framework of wavelets. The simplest wavelet packet $\pi_0(x)$ is the scaling function $\phi(x)$. New wavelet packets $\pi_j(x)$ for $j > 0$ are formed by the recurrence relations

$$\pi_{2j}(x) = \sum_k h_k \pi_j(2x - k)$$

$$\pi_{2j+1}(x) = \sum_k g_k \pi_j(2x - k).$$

where the $h_k$ and $g_k$ are the same as those in the recurrence equations (4.3) and (4.3).

The idea of wavelet packets is most easily seen in the frequency domain. Recall from Figure 14 that each step of the wavelet transform splits the current low frequency subband into two subbands of equal width, one high-pass and one low-pass. With wavelet packets there is a new degree of freedom in the transform. Again there are $N$ stages to the transform for a signal of length $2^N$, but at each stage we have the option of splitting the low-pass subband, the high-pass subband, both, or neither. The high and low pass filters used in each case are the same filters used in the wavelet transform. In fact, the wavelet transform is the special case of a wavelet packet transform in which we always split the low-pass subband. With this increased flexibility we can generate $2^N$ possible different transforms in 1-D. The possible transforms give rise to all possible dyadic partitions of the frequency axis. The increased flexibility does not lead to a large increase in complexity; the worst-case complexity for a wavelet packet transform is $O(N \log N)$.

### 8.2 Frequency Adaptive Coders

The *best basis algorithm* is a fast algorithm for minimizing an additive cost function over the set of all wavelet packet bases. Our analysis of transform coding for Gaussian random processes suggests that we select the basis that maximizes the transform coding gain. The approximation theoretic arguments of Mallat and Falzon [57] suggest that at low bit rates the basis that maximizes the number of coefficients below a given threshold is the best choice. The best basis paradigm can accommodate both of these choices. See [77] for an excellent introduction to wavelet packets and the best basis algorithm. Ramchandran and Vetterli [65] describe an algorithm for finding the best wavelet packet basis for coding a given image using rate-distortion criteria.

An important application of this wavelet-packet transform optimization is the FBI Wavelet/Scalar Quantization Standard for fingerprint compression. The standard uses a wavelet packet decomposition for the transform stage of the encoder [78]. The transform used is fixed for all fingerprints, however, so the FBI coder is a first-generation linear coder.
The benefits of customizing the transform on a per-image basis depend considerably on the image. For the Lena test image the improvement in peak signal to noise ratio is modest, ranging from 0.1 dB at 1 bit per pixel to 0.25 dB at 0.25 bits per pixel. This is because the octave band partitions of the spectrum of the Lena image are nearly flat. The Barbara image (see Figure 16), on the other hand, has a narrow-band peak in the spectrum at high frequencies. Consequently, the PSNR increases by roughly 2 dB over the same range of bitrates [65]. Further impressive gains are obtained by combining the adaptive transform with a zerotree structure [67].

8.3 Space-Frequency Adaptive Coders

The best basis algorithm is not limited only to adaptive segmentation of the frequency domain. Related algorithms permit joint time and frequency segmentations. The simplest of these algorithms performs adapted frequency segmentations over regions of the image selected through a quadtree decomposition procedure [79, 80]. More complicated algorithms provide combinations of spatially varying frequency decompositions and frequency varying spatial decompositions [66]. These jointly adaptive algorithms work particularly well for highly nonstationary images.

The primary disadvantage of these spatially adaptive schemes are that the pre-computation requirements are much greater than for the frequency adaptive coders, and the search is also much larger. A second disadvantage is that both spatial and frequency adaptivity are limited to dyadic partitions. A limitation of this sort is necessary for keeping the complexity manageable, but dyadic partitions are not in general the best ones.

9 Utilizing Intra-band Dependencies

The development of the EZW coder motivated a flurry of activity in the area of zerotree wavelet algorithms. The inherent simplicity of the zerotree data structure, its computational advantages, as well as the potential for generating an embedded bitstream were all very attractive to the coding community. Zerotree algorithms were developed for a variety of applications, and many modifications and enhancements to the algorithm were devised, as described in Section 7.

With all the excitement incited by the discovery of EZW, it is easy to automatically assume that zerotree structures, or more generally inter-band dependencies, should be the focal point of efficient subband image compression algorithms. However, some of the best performing subband image coders known today are not based on zerotrees. In this section, we explore two methods that utilize intra-band dependencies. One of them uses the concept of Trellis Coded Quantization (TCQ). The other uses both
inter- and intra-band information, and is based on a recursive estimation of the variance of the wavelet coefficients. Both of them yield excellent coding results.

9.1 Trellis coded quantization

Trellis Coded Quantization (TCQ) [81] is a fast and effective method of quantizing random variables. Trellis coding exploits correlations between variables. More interestingly, it can use non-rectangular quantizer cells that give it quantization efficiencies not attainable by scalar quantizers. The central ideas of TCQ grew out of the ground-breaking work of Ungerboeck [82] in trellis coded modulation. In this section we describe the operational principles of TCQ, mostly through examples. We will briefly touch upon variations and improvements on the original idea, especially at the low bitrates applicable in image coding. In Section 9.2, we review the use of TCQ in multiresolution image compression algorithms.

The basic idea behind TCQ is the following: Assume that we want to quantize a stationary, memoryless uniform source at the rate of $R$ bits per sample. Performing quantization directly on this uniform source would require an optimum scalar quantizer with $2^N$ reproduction levels (symbols). The idea behind TCQ is to first quantize the source more finely, with $2^{R+k}$ symbols. Of course this would exceed the allocated rate, so we cannot have a free choice of symbols at all times.

In our example we take $k = 1$. The scalar codebook of $2^{R+1}$ symbols is partitioned into subsets of $2^{R-1}$ symbols each, generating four sets. In our example $R = 2$; see Figure 21. The subsets are designed such that each of them represents reproduction points of a coarser, rate-$(R-1)$ quantizer. The four subsets are designated $D_0$, $D_1$, $D_2$, and $D_3$. Also, define $S_0 = D_0 \cup D_2$ and $S_1 = D_1 \cup D_3$, where $S_0$ and $S_1$ are known as supersets.

Obviously, the rate constraint prohibits the specification of an arbitrary symbol out of $2^{R+1}$ symbols. However, it is possible to exactly specify, with $R$ bits, one element out of either $S_0$ or $S_1$. At each sample, assuming we know which one of the supersets to use, one bit can be used to determine the active subset, and $R-1$ bits to specify a codeword from the subset. The choice of superset is determined by the state of a finite state machine, described by a suitable trellis. An example of such a trellis, with eight
states, is given in Figure 22. The subsets \{D_0, D_1, D_2, D_3\} are also used to label the branches of the trellis, so the same bit that specifies the subset (at a given state) also determines the next state of the trellis.

Encoding is achieved by spending one bit per sample on specifying the path through the trellis, while the remaining \(R - 1\) bits specify a codeword out of the active subset. It may seem that we are back to a non-optimal rate-\(R\) quantizer (either \(S_0\) or \(S_1\)). So why all this effort? The answer is that we have more codewords than a rate-\(R\) quantizer, because there is some freedom of choosing from symbols of either \(S_0\) or \(S_1\). Of course this choice is not completely free: the decision made at each sample is linked to decisions made at past and future sample points, through the permissible paths of the trellis. But it is this additional flexibility that leads to the improved performance. Availability of both \(S_0\) and \(S_1\) means that the reproduction levels of the quantizer are, in effect, allowed to “slide around” and fit themselves to the data, subject to the permissible paths on the trellis.

Before we continue with further developments of TCQ and subband coding, we should note that in terms of both efficiency and computational requirements, TCQ is much more similar to VQ than to scalar quantization. Since our entire discussion of transform coding has been motivated by an attempt to avoid VQ, what is the motivation for using TCQ in subband coding, instead of standard VQ? The answer lies in the recursive structure of trellis coding and the existence of a simple dynamic programming method, known as the Viterbi algorithm [83], for finding the TCQ codewords. Although it is true that block quantizers, such as VQ, are asymptotically as efficient as TCQ, the process of approaching the limit is far from trivial for VQ. For a given realization of a random process, the code vectors generated by the VQ of size \(N - 1\) have no clear relationship to those with vector dimension \(N\). In contrast, the trellis encoding algorithm increases
the dimensionality of the problem automatically by increasing the length of the trellis.

The standard version of TCQ is not particularly suitable for image coding, because its performance degrades quickly at low rates. This is due partially to the fact that one bit per sample is used to encode the trellis alone, while interesting rates for image coding are mostly below one bit per sample. Entropy constrained TCQ (ECTCQ) improves the performance of TCQ at low rates. In particular, a version of ECTCQ due to Marcellin [84] addresses two key issues: reducing the rate used to represent the trellis (the so-called “state entropy”), and ensuring that zero can be used as an output codeword with high probability. The codebooks are designed using the algorithm and encoding rule from [85].

9.2 TCQ subband coders

In a remarkable coincidence, at the 1994 International Conference in Image Processing, three research groups [86, 87, 88] presented similar but independently developed image coding algorithms. The main ingredients of the three methods are subband decomposition, classification and optimal rate allocation to different subsets of subband data, and entropy-constrained TCQ. These works have been brought together in [68]. We briefly discuss the main aspects of these algorithms.

Consider a subband decomposition of an image, and assume that the subbands are well represented by a non-stationary random process $X$, whose samples $X_i$ are taken from distributions with variances $\sigma_i^2$. One can compute an “average variance” over the entire random process and perform conventional optimal quantization. But better performance is possible by sending overhead information about the variance of each sample, and quantizing it optimally according to its own p.d.f.

This basic idea was first proposed by Chen and Smith [89] for adaptive quantization of DCT coefficients. In their paper, Chen and Smith proposed to divide all DCT coefficients into four groups according to their “activity level”, i.e. variance, and code each coefficient with an optimal quantizer designed for its group. The question of how to partition coefficients into groups was not addressed, however, and [89] arbitrarily chose to form groups with equal population.\(^9\)

However, one can show that equally populated groups are not always

\(^9\)If for a moment, we disregard the overhead information, the problem of partitioning the coefficients bears a strong resemblance to the problem of best linear transform. Both operations, namely the linear transform and partitioning, conserve energy. The goal in both is to minimize overall distortion through optimal allocation of a finite rate. Not surprisingly, the solution techniques are similar (Lagrange multipliers), and they both generate sets with maximum separation between low and high energies (maximum arithmetic to geometric mean ratio).
a good choice. Suppose that we want to classify the samples into $J$ groups, and that all samples assigned to a given class $i \in \{1, ..., J\}$ are grouped into a source $X_i$. Let the total number of samples assigned to $X_i$ be $N_i$, and the total number of samples in all groups be $N$. Define $p_i = N_i/N$ to be the probability of a sample belonging to the source $X_i$. Encoding the source $X_i$ at rate $R_i$ results in a mean squared error distortion of the form [90]

$$D_i(R_i) = c_i^2 \sigma_i^2 \cdot 2^{-2R_i}$$

(1.33)

where $c_i$ is a constant depending on the shape of the pdf. The rate allocation problem can now be solved using a Lagrange multiplier approach, much in the same way as was shown for optimal linear transforms, resulting in the following optimal rates:

$$R_i = \frac{R}{J} + \frac{1}{2} \log_2 \left( \frac{c_i^2 \sigma_i^2}{\prod_{j=1}^{J} (c_j^2 \sigma_j^2)^{p_j}} \right)$$

(1.34)

where $R$ is the total rate and $R_i$ are the rates assigned to each group. Classification gain is defined as the ratio of the quantization error of the original signal $X$, divided by that of the optimally bit-allocated classified version.

$$G_c = \frac{c^2 \sigma^2}{\prod_{j=1}^{J} (c_j^2 \sigma_j^2)^{p_j}}$$

(1.35)

One aims to maximize this gain over $\{p_i\}$. It is not unexpected that the optimization process can often yield non-uniform $\{p_i\}$, resulting in unequal population of the classification groups. It is noteworthy that non-uniform populations not only have better classification gain in general, but also lower overhead: Compared to a uniform $\{p_i\}$, any other distribution has smaller entropy, which implies smaller side information to specify the classes.

The classification gain is defined for $X_i$ taken from one subband. A generalization of this result in [68] combines it with the conventional coding gain of the subbands. Another refinement takes into account the side information required for classification. The coding algorithm then optimizes the resulting expression to determine the classifications. ECTCQ is then used for final coding.

Practical implementation of this algorithm requires attention to a great many details, for which the interested reader is referred to [68]. For example, the classification maps determine energy levels of the signal, which are related to the location of the edges in the image, and are thus related in different subbands. A variety of methods can be used to reduce the overhead information (in fact, the coder to be discussed in the next section makes the management of side information the focus of its efforts). Other issues include alternative measures for classification, and the usage of arithmetic coded TCQ. The coding results of the ECTCQ based subband coding are some of
the best currently available in the literature, although the computational complexity of these algorithms is also considerably greater than the other methods presented in this paper.

9.3 Mixture Modeling and Estimation

A common thread in successful subband and wavelet image coders is modeling of image subbands as random variables drawn from a mixture of distributions. For each sample, one needs to detect which p.d.f. of the mixture it is drawn from, and then quantize it according to that pdf. Since the decoder needs to know which element of the mixture was used for encoding, many algorithms send side information to the decoder. This side information becomes significant, especially at low bitrates, so that efficient management of it is pivotal to the success of the image coder.

All subband and wavelet coding algorithms discussed so far use this idea in one way or another. They only differ in the constraints they put on side information so that it can be coded efficiently. For example, zerotrees are a clever way of indicating side information. The data is assumed from a mixture of very low energy (zero set) and high energy random variables, and the zero sets are assumed to have a tree structure.

The TCQ subband coders discussed in the last section also use the same idea. Different classes represent different energies in the subbands, and are transmitted as overhead. In [68], several methods are discussed to compress the side information, again based on geometrical constraints on the constituent elements of the mixture (energy classes).

A completely different approach to the problem of handling information overhead is explored in [69, 91]. These two works were developed simultaneously but independently. The version developed in [69] is named Estimation Quantization (EQ) by the authors, and is the one that we present in the following. The title of [91] suggests a focus on entropy coding, but in fact the underlying ideas of the two are remarkably similar. We will refer to the the aggregate class as backward mixture-estimation encoding (BMEE).

BMEE models the wavelet subband coefficients as non-stationary generalized Gaussian, whose non-stationarity is manifested by a slowly varying variance (energy) in each band. Because the energy varies slowly, it can be predicted from causal neighboring coefficients. Therefore, unlike previous methods, BMEE does not send the bulk of mixture information as overhead, but attempts to recover it at the decoder from already transmitted data, hence the designation “backward”. BMEE assumes that the causal neighborhood of a subband coefficient (including parents in a subband tree) has the same energy (variance) as the coefficient itself. The estimate of energy is found by applying a maximum likelihood method to a training set formed by the causal neighborhood.

Similar to other recursive algorithms that involve quantization, BMEE has to contend with the problem of stability and drift. Specifically, the
decoder has access only to quantized coefficients, therefore the estimator of energy at the encoder can only use quantized coefficients. Otherwise, the estimates at the encoder and decoder will vary, resulting in drift problems. This presents the added difficulty of estimating variances from quantized causal coefficients. BMEE incorporates the quantization of the coefficients into the maximum likelihood estimation of the variance.

The quantization itself is performed with a dead-zone uniform quantizer (see Figure 15). This quantizer offers a good approximation to entropy constrained quantization of generalized Gaussian signals. The dead-zone and step sizes of the quantizers are determined through a Lagrange multiplier optimization technique, which was introduced in the section on optimal rate allocation. This optimization is performed offline, once each for a variety of encoding rates and shape parameters, and the results are stored in a look-up table. This approach is to be credited for the speed of the algorithm, because no optimization need take place at the time of encoding the image.

Finally, the backward nature of the algorithm, combined with quantization, presents another challenge. All the elements in the causal neighborhood may sometimes quantize to zero. In that case, the current coefficient will also quantize to zero. This degenerate condition will propagate through the subband, making all coefficients on the causal side of this degeneracy equal to zero. To avoid this condition, BMEE provides for a mechanism to send side information to the receiver, whenever all neighboring elements are zero. This is accomplished by a preliminary pass through the coefficients, where the algorithm tries to "guess" which one of the coefficients will have degenerate neighborhoods, and assembles them to a set. From this set, a generalized Gaussian variance and shape parameter is computed and transmitted to the decoder. Every time a degenerate case happens, the encoder and decoder act based on this extra set of parameters, instead of using the backward estimation mode.

The BMEE coder is very fast, and especially in the low bitrate mode (less than 0.25 bits per pixel) is extremely competitive. This is likely to motivate a re-visitation of the role of side information and the mechanism of its transmission in wavelet coders.

10 Future Trends

Current research in image coding is progressing along a number of fronts. At the most basic level, a new interpretation of the wavelet transform has appeared in the literature. This new theoretical framework, called the lifting scheme [41], provides a simpler and more flexible method for designing wavelets than standard Fourier-based methods. New families of non-separable wavelets constructed using lifting have the potential to improve
coders. One very intriguing avenue for future research is the exploration of the nonlinear analogs of the wavelet transform that lifting makes possible.

The area of classification and backward estimation based coders is an active one. Several research groups are reporting promising results [92, 93].

One very promising research direction is the development of coded images that are robust to channel noise via joint source and channel coding. See for example [94], [95] and [96].

The adoption of wavelet based coding to video signals presents special challenges. One can apply 2-D wavelet coding in combination to temporal prediction (motion estimated prediction), which will be a direct counterpart of current DCT-based video coding methods. It is also possible to consider the video signal as a three-dimensional array of data and attempt to compress it with 3-D wavelet analysis. This approach presents difficulties that arise from the fundamental properties of the discrete wavelet transform. The discrete wavelet transform (as well as any subband decomposition) is a space-varying operator, due to the presence of decimation and interpolation. This space variance is not conducive to compact representation of video signals, as described below.

Video signals are best modeled by 2-D projections whose position in consecutive frames of the video signal varies by unknown amounts. Because vast amounts of information are repeated in this way, one can achieve considerable gain by representing the repeated information only once. This is the basis of motion compensated coding. However, since the wavelet representation of the same 2-D signal will vary once it is shifted\(^{10}\), this redundancy is difficult to reproduce in the wavelet domain. A frequency domain study of the difficulties of 3-D wavelet coding of video is presented in [97], and leads to the same insights. Some attempts have also been made on applying 3-D wavelet coding on the residual 3-D data after motion compensation, but have met with indifferent success.

\(^{10}\)Unless the shift is exactly by a correct multiple of \(M\) samples, where \(M\) is the downsampling rate

11 Summary and Conclusion

Image compression is governed by the general laws of information theory and specifically rate-distortion theory. However, these general laws are non-constructive and the more specific techniques of quantization theory are needed for the actual development of compression algorithms.

Vector quantization can theoretically attain the maximum achievable coding efficiency. However, VQ has three main impediments: computational complexity, delay, and the curse of dimensionality. Transform coding techniques, in conjunction with entropy coding, capture important gains of VQ,
while avoiding most of its difficulties.

Theoretically, the Karhunen-Loéve transform is optimal for Gaussian processes. Approximations to the K-L transform, such as the DCT, have led to very successful image coding algorithms such as JPEG. However, even if one argues that image pixels can be individually Gaussian, they cannot be assumed to be jointly Gaussian, at least not across the image discontinuities. Image discontinuities are the place where traditional coders spend the most rate, and suffer the most distortion. This happens because traditional Fourier-type transforms (e.g., DCT) disperse the energy of discontinuous signals across many coefficients, while the compaction of energy in the transform domain is essential for good coding performance.

The discrete wavelet transform provides an elegant framework for signal representation in which both smooth areas and discontinuities can be represented compactly in the transform domain. This ability comes from the multi-resolution properties of wavelets. One can motivate wavelets through spectral partitioning arguments used in deriving optimal quantizers for Gaussian processes. However, the usefulness of wavelets in compression goes beyond the Gaussian case.

State of the art wavelet coders assume that image data comes from a source with fluctuating variance. Each of these coders provides a mechanism to express the local variance of the wavelet coefficients, and quantizes the coefficients optimally or near-optimally according to that variance. The individual wavelet coders vary in the way they estimate and transmit this variances to the decoder, as well as the strategies for quantizing according to that variance.

Zerotree coders assume a two-state structure for the variances: either negligible (zero) or otherwise. They send side information to the decoder to indicate the positions of the non-zero coefficients. This process yields a non-linear image approximation rather than the linear truncated KLT-based approximation motivated by our Gaussian model. The set of zero coefficients are expressed in terms of wavelet trees (Lewis & Knowles, Shapiro, others) or combinations thereof (Said & Pearlman). The zero sets are transmitted to the receiver as overhead, as well as the rest of the quantized data. Zerotree coders rely strongly on the dependency of data across scales of the wavelet transform.

Frequency-adaptive coders improve upon basic wavelet coders by adapting transforms according to the local inter-pixel correlation structure within an image. Local fluctuations in the correlation structure and in the variance can be addressed by spatially adapting the transform and by augmenting the optimized transforms with a zerotree structure.

Other wavelet coders use dependency of data within the bands (and sometimes across the bands as well). Coders based on Trellis Coded Quantization (TCQ) partition coefficients into a number of groups, according to their energy. For each coefficient, they estimate and/or transmit the group information as well as coding the value of the coefficient with TCQ, ac-
sponding to the nominal variance of the group. Another newly developed class of coders transmit only minimal variance information while achieving impressive coding results, indicating that perhaps the variance information is more redundant than previously thought.

While some of these coders may not employ what might strictly be called a wavelet transform, they all utilize a multi-resolution decomposition, and use concepts that were motivated by wavelet theory. Wavelets and the ideas arising from wavelet analysis have had an indelible effect on the theory and practice of image compression, and are likely to continue their dominant presence in image coding research in the near future.

Acknowledgments: G. Davis thanks the Digital Signal Processing group at Rice University for their generous hospitality during the writing of this paper. This work has been supported in part by a Texas Instruments Visiting Assistant Professorship at Rice University and an NSF Mathematical Sciences Postdoctoral Research Fellowship.

12 References


A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition

LAWRENCE R. RABINER, FELLOW, IEEE

Although initially introduced and studied in the late 1960s and early 1970s, statistical methods of Markov source or hidden Markov modeling have become increasingly popular in the last several years. There are two strong reasons why this has occurred. First the models are very rich in mathematical structure and hence can form the theoretical basis for use in a wide range of applications. Second, the models, when applied properly, work very well in practice for several important applications. In this paper we attempt to carefully and methodically review the theoretical aspects of this type of statistical modeling and show how they have been applied to selected problems in machine recognition of speech.

I. INTRODUCTION

Real-world processes generally produce observable outputs which can be characterized as signals. The signals can be discrete in nature (e.g., characters from a finite alphabet, quantized vectors from a codebook, etc.), or continuous in nature (e.g., speech samples, temperature measurements, music, etc.). The signal source can be stationary (i.e., its statistical properties do not vary with time), or nonstationary (i.e., the signal properties vary over time). The signals can be pure (i.e., coming strictly from a single source), or can be corrupted from other signal sources (e.g., noise) or by transmission distortions, reverberation, etc.

A problem of fundamental interest is characterizing such real-world signals in terms of signal models. There are several reasons why one is interested in applying signal models. First of all, a signal model can provide the basis for a theoretical description of a signal processing system which can be used to process the signal so as to provide a desired output. For example if we are interested in enhancing a speech signal corrupted by noise and transmission distortion, we can use the signal model to design a system which will optimally remove the noise and undo the transmission distortion. A second reason why signal models are important is that they are potentially capable of letting us learn a great deal about the signal source (i.e., the real-world process which produced the signal) without having to have the source available. This property is especially important when the cost of getting signals from the actual source is high.

In this case, with a good signal model, we can simulate the source and learn as much as possible via simulations. Finally, the most important reason why signal models are important is that they often work extremely well in practice, and enable us to realize important practical systems—e.g., prediction systems, recognition systems, identification systems, etc., in a very efficient manner.

These are several possible choices for what type of signal model is used for characterizing the properties of a given signal. Broadly one can dichotomize the types of signal models into the class of deterministic models, and the class of statistical models. Deterministic models generally exploit some known specific properties of the signal, e.g., that the signal is a sine wave, or a sum of exponentials, etc. In these cases, specification of the signal model is generally straightforward; all that is required is to determine (estimate) values of the parameters of the signal model (e.g., amplitude, frequency, phase of a sine wave, amplitudes and rates of exponentials, etc.). The second broad class of signal models is the set of statistical models in which one tries to characterize only the statistical properties of the signal. Examples of such statistical models include Gaussian processes, Poisson processes, Markov processes, and hidden Markov processes, among others. The underlying assumption of the statistical model is that the signal can be well characterized as a parametric random process, and that the parameters of the stochastic process can be determined (estimated) in a precise, well-defined manner.

For the applications of interest, namely speech processing, both deterministic and stochastic signal models have had good success. In this paper we will concern ourselves strictly with one type of stochastic signal model, namely the hidden Markov model (HMM). (These models are referred to as Markov sources or probabilistic functions of Markov chains in the communications literature.) We will first review the theory of Markov chains and then extend the ideas to the class of hidden Markov models using several simple examples. We will then focus our attention on the three fundamental problems for HMM design, namely the

1The idea of characterizing the theoretical aspects of hidden Markov modeling in terms of solving three fundamental problems is due to Jack Ferguson of IDA (Institute for Defense Analysis) who introduced it in lectures and writing.

Manuscript received January 15, 1988; revised October 4, 1988. The author is with AT&T Bell Laboratories, Murray Hill, NJ 07974-2070, USA.
IEEE Log Number 8825949.
evaluation of the probability (or likelihood) of a sequence of observations given a specific HMM; the determination of a best sequence of model states; and the adjustment of model parameters so as to best account for the observed signal. We will show that once these three fundamental problems are solved, we can apply HMMs to selected problems in speech recognition.

Neither the theory of hidden Markov models nor its applications to speech recognition is new. The basic theory was published in a series of classic papers by Baum and his colleagues [1]-[5] in the late 1960s and early 1970s and was implemented for speech processing applications by Baker [6] at CMU, and by Jelinek and his colleagues at IBM [7]-[13] in the 1970s. However, widespread understanding and application of the theory of HMMs to speech processing has occurred only within the past several years. There are several reasons why this has been the case. First, the basic theory of hidden Markov models was published in mathematical journals which were not generally read by engineers working on problems in speech processing. The second reason was that the original applications of the theory to speech processing did not provide sufficient tutorial material for most readers to understand the theory and to be able to apply it to their own research. As a result, several tutorial papers were written which provided a sufficient level of detail for a number of research labs to begin work using HMMs in individual speech processing applications [14]-[19]. This tutorial is intended to provide an overview of the basic theory of HMMs (as originated by Baum and his colleagues), provide practical details on methods of implementation of the theory, and describe a couple of selected applications of the theory to distinct problems in speech recognition. The paper combines results from a number of original sources and hopefully provides a single source for acquiring the background required to pursue further this fascinating area of research.

The organization of this paper is as follows. In Section II we review the theory of discrete Markov chains and show how the concept of hidden states, where the observation is a probabilistic function of the state, can be used effectively. We illustrate the theory with two simple examples, namely coin-tossing, and the classic balls-in-urns system. In Section III we discuss the three fundamental problems of HMMs, and give several practical techniques for solving these problems. In Section IV we discuss the various types of HMMs that have been studied including ergodic as well as left-right models. In this section we also discuss the various model features including the form of the observation density function, the state duration density, and the optimization criterion for choosing optimal HMM parameter values. In Section V we discuss the issues that arise in implementing HMMs including the topics of scaling, initial parameter estimates, model size, model form, missing data, and multiple observation sequences. In Section VI we describe an isolated word speech recognizer, implemented with HMM ideas, and show how it performs as compared to alternative implementations. In Section VII we extend the ideas presented in Section VI to the problem of recognizing a string of spoken words based on concatenating individual HMMs of each word in the vocabulary. In Section VIII we briefly outline how the ideas of HMM have been applied to a large vocabulary speech recognizer, and in Section IX we summarize the ideas discussed throughout the paper.

II. DISCRETE MARKOV PROCESSES

Consider a system which may be described at any time as being in one of a set of N distinct states, S1, S2, · · · , SN, as illustrated in Fig. 1 (where N = 5 for simplicity). At reg-

![Fig. 1. A Markov chain with 5 states (labeled S1 to S5) with selected state transitions.](image)

ularly spaced discrete times, the system undergoes a change of state (possibly back to the same state) according to a set of probabilities associated with the state. We denote the time instants associated with state changes as t = 1, 2, · · · , and we denote the actual state at time t as qt. A full probabilistic description of the above system would, in general, require specification of the current state at time 0, as well as all the predecessor states. For the special case of a discrete, first order, Markov chain, this probabilistic description is truncated to just the current and the predecessor state, i.e.,

\[ P(q_t = S_j | q_{t-1} = S_i, q_{t-2} = S_k, \cdots ) \]

\[ = P(q_t = S_j | q_{t-1} = S_i). \]  

(1)

Furthermore we only consider those processes in which the right-hand side of (1) is independent of time, thereby leading to the set of state transition probabilities aij of the form

\[ a_{ij} = P(q_t = S_j | q_{t-1} = S_i), \quad 1 \leq i,j \leq N \]  

(2)

with the state transition coefficients having the properties

\[ a_{ij} \geq 0 \]  

(3a)

\[ \sum_{j=1}^{N} a_{ij} = 1 \]  

(3b)

since they obey standard stochastic constraints.

The above stochastic process could be called an observable Markov model since the output of the process is the set of states at each instant of time, where each state corresponds to a physical (observable) event. To set ideas, consider a simple 3-state Markov model of the weather. We assume that once a day (e.g., at noon), the weather is

\footnote{A good overview of discrete Markov processes is in [20, ch. 5].}
observed as being one of the following:

State 1: rain or (snow)
State 2: cloudy
State 3: sunny.

We postulate that the weather on day $t$ is characterized by a single one of the three states above, and that the matrix $A$ of state transition probabilities is

$$A = \begin{bmatrix}
0.4 & 0.3 & 0.3 \\
0.2 & 0.6 & 0.2 \\
0.1 & 0.1 & 0.8
\end{bmatrix}.$$

Given that the weather on day 1 ($t = 1$) is sunny (state 3), we can ask the question: What is the probability (according to the model) that the weather for the next 7 days will be “sun-sun-rain-rain-sun-cloudy-sun…”? Stated more formally, we define the observation sequence $O$ as $O = \{S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8\}$ corresponding to $t = 1, 2, \ldots, 8$, and we wish to determine the probability of $O$, given the model. This probability can be expressed (and evaluated) as

$$P(O|\text{Model}) = P(S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8|\text{Model})$$

$$= P(S_1) \cdot P(S_2|S_1) \cdot P(S_3|S_2) \cdot P(S_4|S_3) \cdot P(S_5|S_4) \cdot P(S_6|S_5) \cdot P(S_7|S_6) \cdot P(S_8|S_7)$$

$$= \pi_1 \cdot a_{11} \cdot a_{12} \cdot a_{21} \cdot a_{22} \cdot a_{31} \cdot a_{32} \cdot a_{33} \cdot a_{34} \cdot a_{41} \cdot a_{42} \cdot a_{43} \cdot a_{44} \cdot a_{51} \cdot a_{52} \cdot a_{53} \cdot a_{54} \cdot a_{61} \cdot a_{62} \cdot a_{63} \cdot a_{64} \cdot a_{71} \cdot a_{72} \cdot a_{73} \cdot a_{74} \cdot a_{81} \cdot a_{82} \cdot a_{83} \cdot a_{84}$$

$$= 0.1 \cdot (0.3)(0.6)(0.1)(0.4)(0.3)(0.1)(0.4)(0.2)$$

$$= 1.536 \times 10^{-4}$$

where we use the notation

$$\pi_i = P(q_1 = S_i), \quad 1 \leq i \leq N$$

(4)
to denote the initial state probabilities.

Another interesting question we can ask (and answer using the model) is: Given that the model is in a known state, what is the probability it stays in that state for exactly $d$ days? This probability can be evaluated as the probability of the observation sequence

$$O = \{S_1, S_2, S_3, \ldots, S_d, S_{d+1} \neq S_0\},$$

given the model, which is

$$P(O|\text{Model}, q_1 = S_0) = (a_{0d})^{d-1}(1 - a_{0d}) = p(d).$$

(5)
The quantity $p(d)$ is the (discrete) probability density function of duration $d$ in state $i$. This exponential duration density is characteristic of the state duration in a Markov chain. Based on $p(d)$, we can readily calculate the expected number of observations (duration) in a state, conditioned on starting in that state as

$$\overline{d}_i = \sum_{d=1}^{\infty} dp(d)$$

(6a)

$$= \sum_{d=1}^{\infty} (a_{id})^d(1 - a_{id}) = \frac{1}{1 - a_{id}}.$$ (6b)

Thus the expected number of consecutive days of sunny weather, according to the model, is $1/0.2 = 5$; for cloudy it is $2.5$; for rain it is $1.67$.

A. Extension to Hidden Markov Models

So far we have considered Markov models in which each state corresponded to an observable (physical) event. This model is too restrictive to be applicable to many problems of interest. In this section we extend the concept of Markov models to include the case where the observation is a probabilistic function of the state—i.e., the resulting model (which is called a hidden Markov model) is a doubly embedded stochastic process with an underlying stochastic process that is not observable (it is hidden), but can only be observed through another set of stochastic processes that produce the sequence of observations. To fix ideas, consider the following model of some simple coin tossing experiments.

**Coin Toss Models:** Assume the following scenario. You are in a room with a barrier (e.g., a curtain) through which you cannot see what is happening. On the other side of the barrier is another person who is performing a coin (or multiple coin) tossing experiment. The other person will not tell you anything about what he is doing exactly; he will only tell you the result of each coin flip. Thus a sequence of hidden coin tossing experiments is performed, with the observation sequence consisting of a series of heads and tails; e.g., a typical observation sequence would be

$$O = O_1, O_2, O_3, \ldots, O_T,$$

where $H$ stands for heads and $T$ stands for tails.

Given the above scenario, the problem of interest is how do we build an HMM to explain (model) the observed sequence of heads and tails. The first problem one faces is deciding what the states in the model correspond to, and then deciding how many states should be in the model. One possible choice would be to assume that only a single biased coin was being tossed. In this case we could model the situation with a 2-state model where each state corresponds to a side of the coin (i.e., heads or tails). This model is depicted in Fig. 2(a). In this case the Markov model is observable, and the only issue for complete specification of the model would be to decide on the best value for the bias (i.e., the probability of, say, heads). Interestingly, an equivalent HMM to that of Fig. 2(a) would be a degenerate 1-state model, where the state corresponds to the single biased coin, and the unknown parameter is the bias of the coin.

A second form of HMM for explaining the observed sequence of coin toss outcome is given in Fig. 2(b). In this case there are 2 states in the model and each state corresponds to a different, biased, coin being tossed. Each state is characterized by a probability distribution of heads and tails, and transitions between states are characterized by a state transition matrix. The physical mechanism which accounts for how state transitions are selected could itself be a set of independent coin tosses, or some other probabilistic event.

A third form of HMM for explaining the observed sequence of coin toss outcomes is given in Fig. 2(c). This model corresponds to using 3 biased coins, and choosing from among the three, based on some probabilistic event.
Fig. 2. Three possible Markov models which can account for the results of hidden coin tossing experiments. (a) 1-coin model. (b) 2-coin model. (c) 3-coin model.

Given the choice among the three models shown in Fig. 2 for explaining the observed sequence of heads and tails, a natural question would be which model best matches the actual observations. It should be clear that the simple 1-coin model of Fig. 2(a) has only 1 unknown parameter; the 2-coin model of Fig. 2(b) has 4 unknown parameters; and the 3-coin model of Fig. 2(c) has 9 unknown parameters. Thus, with the greater degrees of freedom, the larger HMMs would seem to inherently be more capable of modeling a series of coin tossing experiments than would equivalently smaller models. Although this is theoretically true, we will see later in this paper that practical considerations impose some strong limitations on the size of models that we can consider. Furthermore, it might just be the case that only a single coin is being tossed. Then using the 3-coin model of Fig. 2(c) would be inappropriate, since the actual physical event would not correspond to the model being used—i.e., we would be using an underspecified system.

The Urn and Ball Model*: To extend the ideas of the HMM to a somewhat more complicated situation, consider the urn and ball system of Fig. 3. We assume that there are $N$ (large) glass urns in a room. Within each urn there are a large number of colored balls. We assume there are $M$ distinct colors of the balls. The physical process for obtaining observations is as follows. A genie is in the room, and according to some random process, he (or she) chooses an initial urn. From this urn, a ball is chosen at random, and its color is recorded as the observation. The ball is then replaced in the urn from which it was selected. A new urn is then selected.

*The urn and ball model was introduced by Jack Ferguson, and his colleagues, in lectures on HMM theory.

Fig. 3. An $N$-state urn and ball model which illustrates the general case of a discrete symbol HMM.

according to the random selection process associated with the current urn, and the ball selection process is repeated. This entire process generates a finite observation sequence of colors, which we would like to model as the observable output of an HMM.

It should be obvious that the simplest HMM that corresponds to the urn and ball process is one in which each state corresponds to a specific urn, and for which a (ball) color probability is defined for each state. The choice of urns is dictated by the state transition matrix of the HMM.

B. Elements of an HMM

The above examples give us a pretty good idea of what an HMM is and how it can be applied to some simple scenarios. We now formally define the elements of an HMM, and explain how the model generates observation sequences.

An HMM is characterized by the following:

1) $N$, the number of states in the model. Although the states are hidden, for many practical applications there is often some physical significance attached to the states or to sets of states of the model. Hence, in the coin tossing experiments, each state corresponded to a distinct biased coin. In the urn and ball model, the states corresponded to the urns. Generally the states are interconnected in such a way that any state can be reached from any other state (e.g., an ergodic model); however, we will see later in this paper that other possible interconnections of states are often of interest. We denote the individual states as $S = \{S_1, S_2, \ldots, S_N\}$, and the state at time $t$ as $q_t$.

2) $M$, the number of distinct observation symbols per state, i.e., the discrete alphabet size. The observation symbols correspond to the physical output of the system being modeled. For the coin toss experiments the observation symbols were simply heads or tails; for the ball and urn model they were the colors of the balls selected from the urns. We denote the individual symbols as $V = \{v_1, v_2, \ldots, v_M\}$.

3) The state transition probability distribution $A = \{a_{ij}\}$ where

$$a_{ij} = P(q_{t+1} = S_j | q_t = S_i), \quad 1 \leq i, j \leq N. \quad (7)$$

For the special case where any state can reach any other state in a single step, we have $a_{ij} > 0$ for all $i, j$. For other types of HMMs, we would have $a_{ij} = 0$ for one or more $(i, j)$ pairs.
4) The observation symbol probability distribution in state \( j \), \( B = \{ b_j(k) \} \), where
\[
b_j(k) = P(v_k, t \mid q_i = s_j), \quad 1 \leq j \leq N
\]
\[
1 \leq k \leq M.
\]
5) The initial state distribution \( \pi = \{ \pi_i \} \) where
\[
\pi_i = P(q_1 = s_i), \quad 1 \leq i \leq N.
\]

Given appropriate values of \( N, M, A, B, \) and \( \pi \), the HMM can be used as a generator to give an observation sequence
\[
O = O_1, O_2, \ldots, O_T
\]
(where each observation \( O_i \) is one of the symbols from \( V \), and \( T \) is the number of observations in the sequence) as follows:

1) Choose an initial state \( q_1 = s_1 \) according to the initial state distribution \( \pi \).
2) Set \( t = 1 \).
3) Choose \( O_1 = v_1 \) according to the symbol probability distribution in state \( s_1 \), i.e., \( b_1(k) \).
4) Transit to a new state \( q_{t+1} = s_t \) according to the state transition probability distribution for state \( s_t \), i.e., \( a_{t-1} \).
5) Set \( t = t + 1 \); return to step 3 if \( t < T \); otherwise terminate the procedure.

The above procedure can be used as both a generator of observations, and as a model for how a given observation sequence was generated by an appropriate HMM.

It can be seen from the above discussion that a specific specification of an HMM requires specification of two model parameters \((N \text{ and } M)\), specification of observation symbols, and the specification of the three probability measures \( A, B, \) and \( \pi \). For convenience, we use the compact notation
\[
\lambda = (A, B, \pi)
\]
to indicate the complete parameter set of the model.

C. The Three Basic Problems for HMMs

Given the form of HMM of the previous section, there are three basic problems of interest that must be solved for the model to be useful in real-world applications. These problems are the following:

- **Problem 1**: Given the observation sequence \( O = O_1, O_2, \ldots, O_T \), and a model \( \lambda = (A, B, \pi) \), how do we efficiently compute \( P(O \mid \lambda) \), the probability of the observation sequence, given the model?

- **Problem 2**: Given the observation sequence \( O = O_1, O_2, \ldots, O_T \), and the model \( \lambda \), how do we choose a corresponding state sequence \( Q = q_1, q_2, \ldots, q_T \) which is optimal in some meaningful sense (i.e., best "explains" the observations)?

- **Problem 3**: How do we adjust the model parameters \( \lambda = (A, B, \pi) \) to maximize \( P(O \mid \lambda) \)?

The material in this section and in Section III is based on the ideas presented by Jack Ferguson of IDA in lectures at Bell Laboratories.
We shall see that the three problems are linked together tightly under our probabilistic framework.

III. SOLUTIONS TO THE THREE BASIC PROBLEMS OF HMMs

A. Solution to Problem 1

We wish to calculate the probability of the observation sequence, \( O = O_1 O_2 \cdots O_T \), given the model \( \lambda \), i.e., \( P(O|\lambda) \). The most straightforward way of doing this is through enumerating every possible state sequence of length \( T \) (the number of observations). Consider one such fixed state sequence

\[
Q = q_1 q_2 \cdots q_T
\]

where \( q_t \) is the initial state. The probability of the observation sequence \( O \) for the state sequence of (12) is

\[
P(O|Q, \lambda) = \prod_{t=1}^{T} P(O_t|q_t, \lambda)
\]

where we have assumed statistical independence of observations. Thus we get

\[
P(O|Q, \lambda) = b_{q_1}(O_1) b_{q_2}(O_2) \cdots b_{q_T}(O_T).
\]

The probability of such a state sequence \( Q \) can be written as

\[
P(Q|\lambda) = \pi_{q_1} q_1 a_{q_1 q_2} q_2 \cdots a_{q_{T-1} q_T}.
\]

The joint probability of \( O \) and \( Q \), i.e., the probability that \( O \) and \( Q \) occur simultaneously, is simply the product of the above two terms, i.e.,

\[
P(O, Q|\lambda) = P(O|Q, \lambda) P(Q|\lambda).
\]

The probability of \( O \) (given the model) is obtained by summing this joint probability over all possible state sequences \( Q \) giving

\[
P(O|\lambda) = \sum_{Q} P(O, Q|\lambda) = \sum_{Q} P(O|Q, \lambda) P(Q|\lambda)
\]

\[
= \sum_{q_1} \pi_{q_1} b_{q_1}(O_1) a_{q_1 q_2} b_{q_2}(O_2) \cdots a_{q_{T-1} q_T} b_{q_T}(O_T).
\]

The interpretation of the computation in the above equation is the following. Initially (at time \( t = 1 \)) we are in state \( q_1 \) with probability \( \pi_{q_1} \), and generate the symbol \( O_1 \) (in this state) with probability \( b_{q_1}(O_1) \). The clock changes from time \( t \) to \( t + 1 \) (\( t = 2 \)) and we make a transition to state \( q_2 \) from state \( q_1 \) with probability \( a_{q_1 q_2} \), and generate symbol \( O_2 \) with probability \( b_{q_2}(O_2) \). This process continues in this manner until we make the last transition (at time \( T \)) from state \( q_{T-1} \) to state \( q_T \) with probability \( a_{q_{T-1} q_T} \) and generate symbol \( O_T \), with probability \( b_{q_T}(O_T) \).

A little thought should convince the reader that the calculation of \( P(O|\lambda) \), according to its direct definition (17) involves on the order of \( 2T \cdot N^T \) calculations, since at every \( t = 1, 2, \cdots, T \), there are \( N \) possible states which can be reached (i.e., there are \( N^T \) possible state sequences), and for each such state sequence about \( 2T \) calculations are required for each term in the sum of (17). (To be precise, we need \( 2T - 1 \) \( N^2 \) multiplications, and \( N^T - 1 \) additions.) This calculation is computationally unfeasible, even for small values of \( N \) and \( T \); e.g., for \( N = 5 \) (states), \( T = 100 \) (observations), there are on the order of \( 2 \cdot 100 \cdot 5^{100} = 10^{22} \) computations! Clearly a more efficient procedure is required to solve Problem 1. Fortunately such a procedure exists and is called the forward–backward procedure.

The Forward–Backward Procedure [2], [3]: Consider the forward variable \( \alpha_t(i) \) defined as

\[
\alpha_t(i) = P(O_t O_{t+1} \cdots O_T | q_t = S_i| \lambda)
\]

i.e., the probability of the partial observation sequence, \( O_t O_{t+1} \cdots O_T \) (until time \( t \)) and state \( S_i \) at time \( t \), given the model \( \lambda \). We can solve for \( \alpha_t(i) \) inductively, as follows:

1) Initialization:

\[
\alpha_t(i) = \pi_{i} b_{i}(O_t), \quad 1 \leq i \leq N.
\]

2) Induction:

\[
\alpha_{t+1}(j) = \sum_{i=1}^{N} \alpha_t(i) a_{ij} b_{j}(O_{t+1}), \quad 1 \leq t \leq T - 1, \quad 1 \leq j \leq N.
\]

3) Termination:

\[
P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i).
\]

Step 1) initializes the forward probabilities as the joint probability of state \( S_i \) and initial observation \( O_t \). The induction step, which is the heart of the forward calculation, is illustrated in Fig. 4(a). This figure shows how state \( S_j \) can be

![Fig. 4. (a) Illustration of the sequence of operations required for the computation of the forward variable \( \alpha_t(i) \). (b) Implementation of the computation of \( \alpha_t(i) \) in terms of a lattice of observations \( t \) and states \( i \).](image)

*Strictly speaking, we only need the forward part of the forward–backward procedure to solve Problem 1. We will introduce the backward part of the procedure in this section since it will be used to help solve Problem 3.*
reached at time $t + 1$ from the $N$ possible states, $S_i$, $1 \leq i \leq N$, at time $t$. Since $a_{ij}(l)$ is the probability of the joint event that $O_1, O_2, \ldots, O_l$ are observed, and the state at time $t$ is $S_j$, the product $a_{ij}(l)q_j$ is then the probability of the joint event that $O_1, O_2, \ldots, O_l$ are observed, and state $S_j$ is reached at time $t + 1$ via state $S_i$ at time $t$. Summing this product over all the $N$ possible states, $S_i$, $1 \leq i \leq N$ at time $t$ results in the probability of $S_j$ at time $t + 1$ with all the accompanying previous partial observations. Once this is done and $S_j$ is known, it is easy to see that $a_{ij}(l)$ is obtained by accounting for observation $O_{l+1}$ in state $j$, i.e., by multiplying the summed quantity by the probability $b_j(O_{l+1})$. The computation of (20) is performed for all states $j$, $1 \leq j \leq N$, for a given $t$; the computation is then iterated for $t = 1, 2, \ldots, T - 1$. Finally, step 3 gives the desired calculation of $P(O|\lambda)$ as the sum of the terminal forward variables $a_{ij}(l)$. This is the case since, by definition,

$$a_{ij}(l) = P(O_1 O_2 \cdots O_l \mid q_j = S_j | \lambda)$$  \hspace{1cm} (22)

and hence $P(O|\lambda)$ is just the sum of the $a_{ij}(l)$'s.

If we examine the computation involved in the calculation of $a_{ij}(l)$, $1 \leq i \leq T$, $1 \leq j \leq N$, we see that it requires on the order of $N^2 T$ calculations, rather than $2TN$ as required by the direct calculation. (Again, to be precise, we need $N(N + 1)(T - 1) + N$ multiplications and $N(N - 1)(T - 1)$ additions.) For $N = 5$, $T = 100$, we need about 3000 computations for the forward method, versus $10^6$ computations for the direct calculation, a savings of about 69 orders of magnitude.

The forward probability calculation is, in effect, based upon the lattice (or trellis) structure shown in Fig. 4(b). The key is that since there are only $N$ states (nodes at each time slot in the lattice), all the possible state sequences will remerge into these $N$ nodes, no matter how long the observation sequence. At time $t = 1$ (the first time slot in the lattice), we need to calculate values of $a_{ij}(1)$, $1 \leq i \leq N$. At times $t = 2, 3, \ldots, T$, we only need to calculate values of $a_{ij}(t)$, $1 \leq j \leq N$, where each calculation involves only $N$ previous values of $a_{il}$ because each of the $N$ grid points is reached from the same $N$ grid points at the previous time slot.

In a similar manner, we can consider a backward variable $\beta_{il}(l)$ defined as

$$\beta_{il}(l) = P(O_{l+1} O_{l+2} \cdots O_N | q_i = S_i | \lambda)$$  \hspace{1cm} (23)

i.e., the probability of the partial observation sequence from $t + 1$ to the end, given state $S_i$ at time $t$ and the model $\lambda$. Again we can solve for $\beta_{il}(l)$ inductively, as follows:

1) Initialization:

$$\beta_{i1}(l) = 1, \hspace{1cm} 1 \leq i \leq N.$$  \hspace{1cm} (24)

2) Induction:

$$\beta_{il}(l) = \sum_{j=1}^{N} a_{ij}(l) \beta_{jl}(l), \hspace{1cm} t = T - 1, T - 2, \ldots, 1, 1 \leq i \leq N.$$  \hspace{1cm} (25)

The initialization step 1 arbitrarily defines $\beta_{i1}(l)$ to be 1 for all $l$. Step 2), which is illustrated in Fig. 5, shows that in order to have been in state $S_i$ at time $t$, and to account for the observation sequence from time $t + 1$ on, you have to consider all possible states $S_i$ at time $t + 1$, accounting for the transition from $S_i$ to $S_j$ (the $a_{ij}$ term), as well as the observation $O_{l+1}$ in state $j$ (the $b_j(O_{l+1})$ term), and then account for the remaining partial observation sequence from state $j$ (the $\beta_{jl}(l)$ term). We will see later how the backward, as well as the forward calculations are used extensively to help solve fundamental Problems 2 and 3 of HMMs.

Again, the computation of $\beta_{il}(l)$, $1 \leq i \leq T, 1 \leq l \leq N$, requires on the order of $N^2 T$ calculations, and can be computed in a lattice structure similar to that of Fig. 4(b).

### B. Solution to Problem 2

Unlike Problem 1 for which an exact solution can be given, there are several possible ways of solving Problem 2, namely finding the “optimal” state sequence associated with the given observation sequence. The difficulty lies with the definition of the optimal state sequence; i.e., there are several possible optimality criteria. For example, one possible optimality criterion is to choose the states $q_i$ which are individually most likely. This optimality criterion maximizes the expected number of correct individual states. To implement this solution to Problem 2, we define the variable

$$\gamma_{il}(l) = P(q_l = S_l | O, \lambda)$$  \hspace{1cm} (26)

i.e., the probability of being in state $S_l$ at time $t$, given the observation sequence $O$, and the model $\lambda$. Equation (26) can be expressed simply in terms of the forward–backward variables, i.e.,

$$\gamma_{il}(l) = \frac{a_{il}(l) \beta_{il}(l)}{P(O|\lambda)} = \frac{a_{il}(l) \beta_{il}(l)}{\sum_{j=1}^{N} a_{ij}(l) \beta_{jl}(l)}$$  \hspace{1cm} (27)

since $a_{il}(l)$ accounts for the partial observation sequence $O_1 O_2 \cdots O_l$ and state $S_l$ at $t$, while $\beta_{il}(l)$ accounts for the remainder of the observation sequence $O_{l+1} O_{l+2} \cdots O_T$, given state $S_l$ at $t$. The normalization factor $P(O|\lambda) = \sum_{i=1}^{N} a_{il}(l) \beta_{il}(l)$ makes $\gamma_{il}(l)$ a probability measure so that

$$\sum_{i=1}^{N} \gamma_{il}(l) = 1.$$  \hspace{1cm} (28)

Using $\gamma_{il}(l)$, we can solve for the individually most likely state $q_t$ at time $t$, as

$$q_t = \operatorname*{argmax}_{1 \leq i \leq N} \left[ \gamma_{il}(l) \right], \hspace{1cm} 1 \leq t \leq T.$$  \hspace{1cm} (29)
Although (29) maximizes the expected number of correct states (by choosing the most likely state for each i), there could be some problems with the resulting state sequence. For example, when the HMM has state transitions which have zero probability (ai = 0 for some i and j), the "optimal" state sequence may, in fact, not even be a valid state sequence. This is due to the fact that the solution of (29) simply determines the most likely state at every instant, without regard to the probability of occurrence of sequences of states.

One possible solution to the above problem is to modify the optimality criterion. For example, one could solve for the state sequence that maximizes the expected number of correct pairs of states (qi, qj+1), or triples of states (qi, qi+1, qi+2), etc. Although these criteria might be reasonable for some applications, the most widely used criterion is to find the single best state sequence (path), i.e., to maximize P(Q | O, λ) which is equivalent to maximizing P(O | Q, λ). A formal technique for finding this single best state sequence exists, based on dynamic programming methods, and is called the Viterbi algorithm.

**Viterbi Algorithm** [21, 22]: To find the single best state sequence, Q = {q1, q2, ..., qT}, for the given observation sequence O = {O1, O2, ..., OT}, we need to define the quantity

\[ δ(i) = \max_{q_1, q_2, ..., q_{i-1}} P(q_1, q_2, ..., q_i | i, O_1, O_2, ..., O_i, \lambda) \]

(30)

i.e., δ(i) is the best score (highest probability) along a single path, at time i, which accounts for the first i observations and ends in state Qi. By induction we have

\[ δ(i, j) = [\max_{1 \leq i \leq N} δ(i-1, a_i)] \cdot b_j(O_{i+1}, \lambda). \]

(31)

To actually retrieve the state sequence, we need to keep track of the argument which maximized (31), for each i and j. We do this via the array Ψ(j). The complete procedure for finding the best state sequence can now be stated as follows:

1) Initialization:

\[ δ(0, i) = π_i b_i(O_1), \quad 1 \leq i \leq N \]

(32a)

\[ \Psi(0, i) = 0. \]

(32b)

2) Recursion:

\[ δ(i) = \max_{1 \leq i \leq N} [\delta(i-1, a_i) b_i(O_i)], \quad 2 \leq t \leq T \]

(33a)

\[ \Psi(i, j) = \arg \max_{1 \leq i \leq N} [\delta(i-1, a_i) b_j(O_{i+1})], \quad 2 \leq t \leq T, \quad 1 \leq j \leq N. \]

(33b)

3) Termination:

\[ P^* = \max_{1 \leq i \leq N} δ(T, i) \]

(34a)

\[ q_1^* = \arg \max_{1 \leq i \leq N} δ(T, i). \]

(34b)

4) Path (state sequence) backtracking:

\[ q_1^* = \Psi(T, q_1^*), \quad t = T-1, T-2, \ldots, 1. \]

(35)

It should be noted that the Viterbi algorithm is similar (except for the backtracking step) in implementation to the forward calculation of (19)-(21). The major difference is the maximization in (33a) over previous states which is used in place of the summing procedure in (20). It also should be clear that a lattice (or trellis) structure efficiently implements the computation of the Viterbi procedure.

**C. Solution to Problem 3 [1]-[5]**

The third, and by far the most difficult, problem of HMMs is to determine a method to adjust the model parameters (A, B, π) to maximize the probability of the observation sequence given the model. There is no known way to analytically solve for the model which maximizes the probability of the observation sequence. In fact, given any finite observation sequence as training data, there is no optimal way of estimating the model parameters. We can, however, choose λ = (A, B, π) such that P(O | λ) is locally maximized using an iterative procedure such as the Baum-Welch method (or equivalently the EM (expectation-maximization) method [23]), or using gradient techniques [14]. In this section we discuss one iterative procedure, based primarily on the classic work of Baum and his colleagues, for choosing model parameters.

In order to describe the procedure for reestimation (iterative update and improvement) of HMM parameters, we first define ξ(i, j), the probability of being in state Si at time ti and state Sj at time ti + 1, given the model and the observation sequence, i.e.

\[ ξ(i, j) = P(q_i = S_i, q_{i+1} = S_j | O_i, λ). \]

(36)

The sequence of events leading to the conditions required by (36) is illustrated in Fig. 6. It should be clear, from the definitions of the forward and backward variables, that we can write ξ(i, j) in the form

\[ ξ(i, j) = \frac{α_i b_j(O_{i+1}) β_{i+1}(j)}{P(O | λ)} \]

(37)

where the numerator term is just P(q_i = S_i, q_{i+1} = S_j | O_i, λ) and the division by P(O | λ) gives the desired probability measure.
We have previously defined $\gamma(i(t))$ as the probability of being in state $S_i$ at time $t$, given the observation sequence and the model; hence we can relate $\gamma(i(t))$ to $\xi(i, j)$ by summing over $j$, giving

$$\gamma(i(t)) = \sum_{j=1}^{N} \xi(i, j).$$  \hspace{1cm} (38)

If we sum $\gamma(i(t))$ over the time index $t$, we get a quantity which can be interpreted as the expected (over time) number of times that state $S_i$ is visited, or equivalently, the expected number of transitions made from state $S_i$ (if we exclude the time slot $t$ from the summation). Similarly, summation of $\xi(i, j)$ over $t$ (from $t = 1$ to $t = T - 1$) can be interpreted as the expected number of transitions from state $S_i$ to state $S_j$. That is

$$\sum_{t=1}^{T-1} \gamma(i(t)) = \text{expected number of transitions from } S_i \text{ to } S_j$$  \hspace{1cm} (39a)

$$\sum_{t=1}^{T-1} \xi(i, j) = \text{expected number of transitions from } S_i \text{ to } S_j.$$  \hspace{1cm} (39b)

Using the above formulas (and the concept of counting event occurrences) we can give a method for reestimation of the parameters of an HMM. A set of reasonable reestimation formulas for $\pi$, $A$, and $B$ are

$$\bar{\pi}_i = \frac{\text{expected frequency (number of times) in state } S_i \text{ at } (t = 1) = \gamma(i)}{\sum_{i=1}^{N} \gamma(i)}$$  \hspace{1cm} (40a)

$$\bar{a}_{ij} = \frac{\text{expected number of transitions from state } S_i \text{ to state } S_j}{\sum_{i=1}^{T-1} \gamma(i)}$$  \hspace{1cm} (40b)

$$\bar{b}_j(k) = \frac{\text{expected number of times in state } j \text{ and observing symbol } v_k}{\sum_{i=1}^{T} \gamma(i)}$$  \hspace{1cm} (40c)

If we define the current model as $\lambda = (A, B, \pi)$, and use that to compute the right-hand sides of (40a)–(40c), and we define the reestimated model as $\bar{\lambda} = \bar{A}, \bar{B}, \bar{\pi}$, as determined from the left-hand sides of (40a)–(40c), then it has been proven by Baum and his colleagues [6, 3] that either 1) the initial model $\lambda$ defines a critical point of the likelihood function, in which case $\bar{\lambda} = \lambda$; or 2) model $\bar{\lambda}$ is more likely than model $\lambda$ in the sense that $P(O(\bar{\lambda})) > P(O(\lambda))$, i.e., we have found a new model $\bar{\lambda}$ from which the observation sequence is more likely to have been produced.

Based on the above procedure, if we iteratively use $\bar{\lambda}$ in place of $\lambda$ and repeat the reestimation calculation, we then can improve the probability of $O$ being observed from the model until some limiting point is reached. The final result of this reestimation procedure is called a maximum likelihood estimate of the HMM. It should be pointed out that the forward–backward algorithm leads to local maxima only, and that in most problems of interest, the optimization surface is very complex and has many local maxima.

The reestimation formulas of (40a)–(40c) can be derived directly by maximizing (using standard constrained optimization techniques) Baum’s auxiliary function

$$Q(\lambda, \bar{\lambda}) = \sum_{O} P(O, \lambda) \log \left[ P(O, Q(\lambda)) \right]$$  \hspace{1cm} (41)

over $\bar{\lambda}$. It has been proven by Baum and his colleagues [6, 3] that maximization of $Q(\lambda, \bar{\lambda})$ leads to increased likelihood, i.e.

$$\max_{\lambda} \left[ Q(\lambda, \bar{\lambda}) \right] = P(O(\bar{\lambda}) \geq P(O(\lambda)).$$  \hspace{1cm} (42)

Eventually the likelihood function converges to a critical point.

Notes on the Reestimation Procedure: The reestimation formulas can readily be interpreted as an implementation of the EM algorithm of statistics [23] in which the E (expectation) step is the calculation of the auxiliary function $Q(\lambda, \bar{\lambda})$, and the M (modification) step is the maximization over $\bar{\lambda}$. Thus the Baum–Welch reestimation equations are essentially identical to the EM steps for this particular problem.

An important aspect of the reestimation procedure is that the stochastic constraints of the HMM parameters, namely

$$\sum_{i=1}^{N} \bar{\pi}_i = 1$$  \hspace{1cm} (43a)

$$\sum_{i=1}^{N} \bar{a}_{ij} = 1, \quad 1 \leq i \leq N$$  \hspace{1cm} (43b)

$$\sum_{k=1}^{M} \bar{b}_j(k) = 1, \quad 1 \leq j \leq N$$  \hspace{1cm} (43c)

are automatically satisfied at each iteration. By looking at the parameter estimation problem as a constrained optimization of $P(O(\lambda))$ (subject to the constraints of (43)), the techniques of Lagrange multipliers can be used to find the values of $\pi, a_{ij}$, and $b_j(k)$ which maximize $P$ (we use the notation $P = P(O(\lambda))$ as shorthand in this section). Based on setting up a standard Lagrange optimization using Lagrange multipliers, it can readily be shown that $P$ is maximized when
the following conditions are met:

\[ \pi_i = \frac{\partial P}{\partial \pi_i} \sum_{k=1}^{N} \pi_k \frac{\partial P}{\partial \pi_k} \]  \hspace{1cm} (44a)

\[ a_{ij} = \frac{\partial P}{\partial a_{ij}} \sum_{k=1}^{N} \pi_k \frac{\partial P}{\partial a_{ik}} \]  \hspace{1cm} (44b)

\[ b_j(k) = \frac{\partial P}{\partial b_j(k)} \sum_{l=1}^{K} b_l(k) \frac{\partial P}{\partial b_l(l)} \]  \hspace{1cm} (44c)

By appropriate manipulation of (44), the right-hand sides of each equation can be readily converted to be identical to the right-hand sides of each part of (40a)-(40c), thereby showing that the reestimation formulas are indeed exactly correct at critical points of \(P\). In fact the form of (44) is essentially that of a reestimation formula in which the left-hand side is the reestimate and the right-hand side is computed using the current values of the variables.

Finally, we note that since the entire problem can be set up as an optimization problem, standard gradient techniques can be used to solve for "optimal" values of the model parameters [14]. Such procedures have been tried and have been shown to yield solutions comparable to those of the standard reestimation procedures.

IV. TYPES OF HMMs

Until now, we have only considered the special case of ergodic or fully connected HMMs in which every state of the model could be reached (in a single step) from every other state of the model. (Strictly speaking, an ergodic model has the property that every state can be reached from every other state in a finite number of steps.) As shown in Fig. 7(a), for an \(N = 4\) state model, this type of model has the property that every \(a_{ij}\) coefficient is positive. Hence for the example of Fig. 7a we have

\[ A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \]

For some applications, in particular those to be discussed later in this paper, other types of HMMs have been found to account for observed properties of the signal being modeled better than the standard ergodic model. One such model is shown in Fig. 7(b). This model is called a left-right model or a Bakis model [11], [10] because the underlying state sequence associated with the model has the property that as time increases the state index increases (or stays the same), i.e., the states proceed from left to right. Clearly the left-right type of HMM has the desirable property that it can readily model signals whose properties change over time—e.g., speech. The fundamental property of all left-right

HMMs is that the state transition coefficients have the property

\[ a_{ij} = 0, \quad j < i \]  \hspace{1cm} (45)

i.e., no transitions are allowed to states whose indices are lower than the current state. Furthermore, the initial state probabilities have the property

\[ \pi_i = \begin{cases} 0, & i \neq 1 \\ 1, & i = 1 \end{cases} \]  \hspace{1cm} (46)

since the state sequence must begin in state 1 and end in state \(N\). Often, with left-right models, additional constraints are placed on the state transition coefficients to make sure that large changes in state indices do not occur; hence a constraint of the form

\[ a_{ij} = 0, \quad j > i + \Delta \]  \hspace{1cm} (47)

is often used. In particular, for the example of Fig. 7(b), the value of \(\Delta\) is 2, i.e., no jumps of more than 2 states are allowed. The form of the state transition matrix for the example of Fig. 7(b) is thus

\[ A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{bmatrix} \]

It should be clear that, for the last state in a left-right model, that the state transition coefficients are specified as

\[ a_{N1} = 1 \]  \hspace{1cm} (48a)

\[ a_{Ni} = 0, \quad i < N. \]  \hspace{1cm} (48b)
Although we have dichotomized HMMs into ergodic and left-right models, there are many possible variations and combinations possible. By way of example, Fig. 7(c) shows a cross-coupled connection of two parallel left-right HMMs. Strictly speaking, this model is a left-right model (it obeys all the a_k constraints); however, it can be seen that it has certain flexibility not present in a strict left-right model (i.e., one without parallel paths).

It should be clear that the imposition of the constraints of the left-right model, or those of the constrained jump model, essentially have no effect on the reestimation procedure. This is the case because any HMM parameter set to zero initially, will remain at zero throughout the reestimation procedure (see (44)).

A. Continuous Observation Densities in HMMs [24]-[26]

All of our discussion, to this point, has considered only the case when the observations were characterized as discrete symbols chosen from a finite alphabet, and therefore we could use a discrete probability density within each state of this model. The problem with this approach, at least for some applications, is that the observations are continuous signals (or vectors). Although it is possible to quantize such continuous signals via codebooks, etc., there might be serious degradation associated with such quantization. Hence it would be advantageous to be able to use HMMs with continuous observation densities.

In order to use a continuous observation density, some restrictions have to be placed on the form of the model probability density function (pdf) to insure that the parameters of the pdf can be reestimated in a consistent way. The most general representation of the pdf, for which a reestimation procedure has been formulated [24]-[26], is a finite mixture of the form

\[
b_i(O) = \sum_{m=1}^{M} c_{jm} \delta(O, \mu_{jm}, U_{jm}), \quad 1 \leq j \leq N \tag{49}\]

where \(O\) is the vector being modeled, \(c_{jm}\) is the mixture coefficient for the \(m\)th mixture in state \(j\) and \(\delta\) is any log-concave or elliptically symmetric density [24] (e.g., Gaussian), with mean vector \(\mu_{jm}\) and covariance matrix \(U_{jm}\) for the \(m\)th mixture component in state \(j\). Usually a Gaussian density is used for \(\delta\). The mixture gains \(c_{jm}\) satisfy the stochastic constraint

\[
\sum_{m=1}^{M} c_{jm} = 1, \quad 1 \leq j \leq N \tag{50a}\]

\[
c_{jm} \geq 0, \quad 1 \leq j \leq N, 1 \leq m \leq M \tag{50b}\]

so that the pdf is properly normalized, i.e.,

\[
\int_{-\infty}^{\infty} b_i(O) dO = 1, \quad 1 \leq j \leq N. \tag{51}\]

The pdf of (49) can be used to approximate, arbitrarily closely, any finite, continuous density function. Hence it can be applied to a wide range of problems.

It can be shown [24]-[26] that the reestimation formulas for the coefficients of the mixture density, i.e., \(c_{jm}, \mu_{jm}, U_{jm}\), are of the form

\[
\bar{\gamma}_{jk} = \frac{\sum_{i=1}^{T} \gamma(j,k) \cdot O_i}{\sum_{i=1}^{T} \gamma(j,k)} \tag{52}\]

\[
\bar{\mu}_{jk} = \frac{\sum_{i=1}^{T} \gamma(j,k) \cdot O_i}{\sum_{i=1}^{T} \gamma(j,k)} \tag{53}\]

\[
U_{jk} = \frac{\sum_{i=1}^{T} \gamma(j,k) \cdot (O_i - \mu_{jk})(O_i - \mu_{jk})'}{\sum_{i=1}^{T} \gamma(j,k)} \tag{54}\]

where prime denotes transpose and where \(\gamma(j,k)\) is the probability of being in state \(j\) at time \(t\) with the \(k\)th mixture component accounting for \(O_i\), i.e.,

\[
\gamma(j,k) = \frac{c_{jk} \delta(O_t, \mu_{jk}, U_{jk})}{\sum_{r=1}^{N} c_{jr} \delta(O_t, \mu_{jr}, U_{jr})}. \tag{55}\]

(The term \(\gamma(j,k)\) generalizes to \(\gamma(j)\) of (26) in the case of a simple mixture, or a discrete density.) The reestimation formula for \(a_{jk}\) is identical to the one used for discrete observation densities (i.e., (40b)). The interpretation of (52)-(54) is fairly straightforward. The reestimation formula for \(c_{jk}\) is the ratio between the expected number of times the system is in state \(j\) using the \(k\)th mixture component, and the expected number of times the system is in state \(j\). Similarly, the reestimation formula for the mean vector \(\mu_{jk}\) weights each number term of (52) by the observation, thereby giving the expected value of the portion of the observation vector accounted for by the \(k\)th mixture component. A similar interpretation can be given for the reestimation term for the covariance matrix \(U_{jk}\).

B. Autoregressive HMMs [27], [28]

Although the general formulation of continuous density HMMs is applicable to a wide range of problems, there is one other very interesting class of HMMs that is particularly applicable to speech processing. This is the class of autoregressive HMMs [27], [28]. For this class, the observation vectors are drawn from an autoregression process.

To be more specific, consider the observation vector \(O\) with components \(x_0, x_1, x_2, \cdots, x_{K-1}\). Since the basis probability density function for the observation vector is Gaussian autoregressive (or order \(p\)), then the components of \(O\) are related by

\[
O_k = -p \sum_{i=1}^{p} a_i O_{k-i} + e_k \tag{55}\]

where \(e_k, k = 0, 1, 2, \cdots, K-1\) are Gaussian, independent, identically distributed random variables with zero mean and variance \(\sigma^2\), and \(a_i, i = 1, 2, \cdots, p\), are the autoregression or predictor coefficients. It can be shown that for large \(K\), the density function for \(O\) is approximately

\[
f(O) = (2\pi\sigma^2)^{-K/2} \exp \left\{ -\frac{1}{2\sigma^2} \delta(O, a) \right\} \tag{56}\]

where

\[
\delta(O, a) = r(0) r(0) + 2 \sum_{i=1}^{p} r(i) r(i) \tag{57a}\]

\[
a' = [1, a_1, a_2, \cdots, a_p] \tag{57b}\]
\[ t_i = \sum_{n=0}^{p-i} a_{n0} r_{i+n} \quad (a_0 = 1), \quad 1 \leq i \leq p \quad (57c) \]

\[ r(i) = \sum_{n=0}^{K-i} x_n x_{n+i} \quad 0 \leq i \leq p. \quad (57d) \]

In the above equations it can be recognized that \( t(i) \) is the autocorrelation of the observation samples, and \( r(i) \) is the autocorrelation of the autoregressive coefficients.

The total (frame) prediction residual \( \alpha \) can be written as

\[ \alpha = \text{E} \left[ \sum_{i=1}^{K} (e^2)^{i} \right] = K \sigma^2 \quad (58) \]

where \( \sigma^2 \) is the variance per sample of the error signal. Consider the normalized observation vector

\[ \hat{O} = \frac{O}{\sqrt{\sigma^2}} = \frac{O}{\sqrt{K} \sigma} \quad (59) \]

where each sample \( x_i \) is divided by \( \sqrt{K} \sigma \), i.e., each sample is normalized by the sample variance. Then \( \Gamma(\hat{O}) \) can be written as

\[ f(\hat{O}) = \left( \frac{2\pi}{K} \right)^{-K/2} \exp \left( -\frac{K}{2} \hat{O}(\hat{O}, a) \right). \quad (60) \]

In practice, the factor \( K \) (in front of the exponential of (60)) is replaced by an effective frame length \( \hat{K} \) which represents the effective length of each data vector. Thus if consecutive data vectors are overlapped by 3 to 1, then we would use \( \hat{K} = K/3 \) in (60), so that the contribution of each sample of signal to the overall density is counted exactly once.

The way in which we use Gaussian autoregressive density in HMMs is straightforward. We assume a mixture density of the form

\[ b_m(O) = \sum_{m=1}^{M} c_m b_m(O) \quad (61) \]

where each \( b_m(O) \) is the density defined by (60) with autoregression vector \( a_m \) (or equivalently by autocorrelation vector \( r_{a_m} \)), i.e.,

\[ b_m(O) = \left( \frac{2\pi}{K} \right)^{-K/2} \exp \left( -\frac{K}{2} \hat{O}(\hat{O}, a_m) \right). \quad (62) \]

A reestimation formula for the sequence autocorrelation, \( t(i) \) of (57d), for the \( j \)th state, \( k \)th mixture, component has been derived, and is of the form

\[ r_{jk} = \frac{\sum_{i=t}^{T} \gamma(j, k) \cdot r_i}{\sum_{i=t}^{T} \gamma(j, k)} \quad (63a) \]

where \( \gamma(j, k) \) is defined as the probability of being in state \( j \) at time \( t \) and using mixture component \( k \), i.e.,

\[ \gamma(j, k) = \left[ \frac{c_{jk} b_j(O)}{\sum_{k=1}^{N} c_{jk} b_j(O)} \right] \quad (63b) \]

It can be seen that \( r_{jk} \) is a weighted sum (by probability of occurrence) of the normalized autocorrelations of the frames in the observation sequence. From \( r_{jk} \), one can solve a set of normal equations to obtain the corresponding autoregressive coefficient vector \( \hat{a}_{jk} \), for the \( k \)th mixture of state \( j \). The new autocorrelation vectors of the autoregression coefficients can then be calculated using (57c), thereby closing the reestimation loop.

C. Variants on HMM Structures—Null Transitions and Tied States

Throughout this paper we have considered HMMs in which the observations were associated with states of the model. It is also possible to consider models in which the observations are associated with the arcs of the model. This type of HMM has been used extensively in the IBM continuous speech recognizer [13]. It has been found useful, for this type of model, to allow transitions which produce no output—i.e., jumps from one state to another which produce no observation [13]. Such transitions are called null transitions and are designated by a dashed line with the symbol \( \phi \) used to denote the null output.

Fig. 8 illustrates 3 examples (from speech processing tasks) where null arcs have been successfully utilized. The example of part (a) corresponds to an HMM (a left-right model) with a large number of states in which it is possible to omit transitions between any pair of states. Hence it is possible to generate observation sequences with as few as 1 observation and still account for a path which begins in state 1 and ends in state \( N \).

The example of Fig. 8(b) is a finite state network (FSN) representation of a word in terms of linguistic units (i.e., the sound on each arc is itself an HMM). For this model the null transition gives a compact and efficient way of describing alternate word pronunciations (i.e., symbol deletions).

Finally the FSN of Fig. 8(c) shows how the ability to insert a null transition into a grammar network allows a relatively simple network to generate arbitrarily long word (digit) sequences. In the example shown in Fig. 8(c), the null transition allows the network to generate arbitrary sequences of digits of arbitrary length by returning to the initial state after each individual digit is produced.

Another interesting variation in the HMM structure is the concept of parameter tying [13]. Basically the idea is to set up an equivalence relation between HMM parameters in
different states. In this manner the number of independent parameters in the model is reduced and the parameter estimation becomes somewhat simpler. Parameter tying is used in cases where the observation density (for example) is known to be the same in $2$ or more states. Such cases occur often in characterizing speech sounds. The technique is especially appropriate in the case where there is insufficient training data to estimate, reliably, a large number of model parameters. For such cases it is appropriate to tie model parameters so as to reduce the number of parameters (i.e., size of the model) thereby making the parameter estimation problem somewhat simpler. We will discuss this method later in this paper.

**D. Inclusion of Explicit State Duration Density in HMMS**

Perhaps the major weakness of conventional HMMS is the modeling of state duration. Earlier we showed (5) that the inherent duration probability density $p(d)$ associated with state $S_j$ with self transition coefficient $a_j$, was of the form

$$p(d) = (1 - a_j)^{d - 1} a_j$$

(probability of $d$ consecutive observations in state $S_j$)  

(64)

For most physical signals, this exponential state duration density is inappropriate. Instead we would prefer to explicitly model duration density in some analytic form. Fig. 9 illustrates, for a pair of model states $S_i$ and $S_j$, the differences between HMMS without and with explicit duration density. In part (a) the states have exponential duration densities based on self-transition coefficients $a_i$ and $a_j$, respectively. In part (b), the self-transition coefficients are set to zero, and an explicit duration density is specified. For this case, a transition is made only after the appropriate number of observations have occurred in the state (as specified by the duration density).

Based on the simple model of Fig. 9(b), the sequence of events of the variable duration HMM is as follows:

1) An initial state, $q_1 = S_j$ is chosen according to the initial state distribution $\pi_j$.

2) A duration $d_1$ is chosen according to the state duration density $p_0(d_1)$. (For expedience and ease of implementation the duration density $p_0(d)$ is truncated at a maximum duration value $D_0$.)

3) Observations $O_1, O_2, \cdots, O_{d_1}$ are chosen according to the joint observation density, $b_{ij}(O_1, O_2, \cdots , O_{d_1})$. Generally we assume independent observations so that $b_{ij}(O_1, O_2, \cdots , O_{d_1}) = \Pi_{s_i} b_{ij}(O_s)$. The next state, $q_2 = S_j$, is chosen according to the state transition probabilities, $a_{ij}$, with the constraint that $a_{i,j} = 0$, i.e., no transition back to the same state can occur. (Clearly this is a requirement since we assume that, in state $q_1$, exactly $d_1$ observations occurred.)

A little thought should convince the reader that the variable duration HMM can be made equivalent to the standard HMM by setting $p(d)$ to be the exponential density of (64).

Using the above formulation, several changes must be made to the formulas of Section III to allow calculation of $P(O|\lambda)$ and for reestimation of all model parameters. In particular we assume that the first state begins at $t = 1$ and the last state ends at $t = T$, i.e., entire duration intervals are included with the observation sequence. We then define the forward variable $\alpha_i(t)$ as

$$\alpha_i(t) = P(O_1, O_2, \cdots, O_t, S_i \text{ ends at } t|\lambda).$$

(65)

We assume that a total of $r$ states have been visited during the first $t$ observations and we denote the states as $q_1, q_2, \cdots, q_r$ with durations associated with each state of $d_1, d_2, \cdots, d_r$. Thus the constraints of (65) are

$$q_r = S_i,$$

(66a)

$$\sum_{s=1}^{r} d_s = t.$$  

(66b)

Equation (65) can then be written as

$$\alpha_i(t) = \sum_{q} \sum_{d} P_i(q_1) \cdot p_0(d_1) \cdot P(O_1, O_2, \cdots, O_{d_1}, q_1) \cdot a_{q_1,q_2} P_2(d_2) \cdot P(O_{d_1+1}, O_{d_1+2}, \cdots, O_{d_1+d_2}, q_2) \cdots$$

$$\cdot a_{q_{r-1},q_r} P_r(d_r) \cdot P(O_{d_1+\cdots+d_{r-1}+1}, \cdots, O_t, q_r).$$

(67)

where the sum is over all states $q$ and all possible state durations $d$. By induction we can write $\alpha_i(t)$ as

$$\alpha_i(t) = \sum_{j=1}^{N} \sum_{d=1}^{D} \alpha_{i-d} \cdot a_{i-j} \cdot p(j) \cdot P(O_{d_1+\cdots+d_{i-j+1}}, \cdots, O_t) \cdot b_j(q)$$

(68)

where $D$ is the maximum duration within any state. To initialize the computation of $\alpha_i(t)$ we use

$$\alpha_i(t) = \pi_i \cdot p(1) \cdot b_i(q_1)$$

(69a)

$$\alpha_i(t) = \pi_i \cdot p(2) \cdot b_i(q_1) + \sum_{j=1}^{N} \frac{\alpha_i(j)}{\pi_i} \cdot \alpha_{i-j}(1) \cdot b_j(q_2)$$

(69b)
\[ \alpha_t(i) = \pi_t p_t(3) \prod_{i=1}^{3} b_i(O_3) + \frac{2}{\sum_{d=1}^{N} \sum_{j=1}^{i} \alpha_{t-1,j}) \cdot a_{i} \cdot p_{t}(d) \]
\[ \prod_{s=t+1-d}^{t} b_s(O_3) \] (69c)

etc., until \( \alpha_t(i) \) is computed; then (68) can be used for all \( t > D \). It should be clear that the desired probability of \( O \) given the model \( \lambda \) can be written in terms of the \( \alpha_s \) as

\[ P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \] (70)

as was previously used for ordinary HMMs.

In order to give reestimation formulas for all the variables of the variable duration HMM, we must define three more forward-backward variables, namely

\[ \alpha_t^*(i) = P(O_1 O_2 \cdots O_t, S_t \text{ begins at } t+1|\lambda) \] (71)
\[ \beta_t(i) = P(O_{t+1} \cdots O_T|S_t \text{ ends at } t, \lambda) \] (72)
\[ \beta_t^*(i) = P(O_{t+1} \cdots O_T|S_t \text{ begins at } t+1, \lambda) \] (73)

The relationships between \( \alpha, \alpha^*, \beta, \) and \( \beta^* \) are as follows:

\[ \alpha_t^*(i) = \sum_{j=1}^{N} \alpha_t(i) a_{i} \beta_t^*(j) \]
\[ \alpha_t(i) = \sum_{d=1}^{D} \alpha_t^*(i) a_{i} \cdot p_{t}(d) \prod_{s=t+1-d}^{t} b_{s}(O_3) \] (75)
\[ \beta_t(i) = \sum_{j=1}^{N} \beta_t(i) a_{j} \beta_t^*(j) \]
\[ \beta_t^*(i) = \sum_{d=1}^{D} \beta_t^*(i) a_{i} \cdot p_{t}(d) \prod_{s=t+1}^{t+d} b_{s}(O_3) \] (77)

Based on the above relationships and definitions, the reestimation formulas for the variable duration HMM are

\[ \overline{\alpha}_i = \frac{\pi_t p_t(i)}{P(O|\lambda)} \] (78)
\[ \overline{a}_{i} = \frac{\sum_{t=1}^{T} \alpha_t(i) a_{i} \beta_t^*(j)}{\sum_{j=1}^{N} \sum_{t=1}^{T} \alpha_t(i) a_{i} \beta_t^*(j)} \] (79)
\[ \overline{b}_{i}(k) = \frac{\sum_{t=1}^{T} \sum_{s=1}^{\text{all } O_t = k} \left[ \alpha_t(i) \cdot \beta_t^*(i) - \sum_{r < t} \alpha_t(r) \beta_r(i) \right]}{\sum_{t=1}^{T} \sum_{s=1}^{\text{all } O_t = k} \left[ \sum_{r < t} \alpha_t(r) \cdot \beta_t^*(i) - \sum_{r < t} \alpha_t(r) \beta_r(i) \right]} \] (80)
\[ \overline{p}_{t}(d) = \frac{\sum_{t=1}^{T} \alpha_t(i) \cdot p_{t}(d) \cdot \beta_t^*(i) \prod_{s=t+1-d}^{t} b_{s}(O_3)}{\sum_{t=1}^{T} \sum_{s=1}^{\text{all } O_t = k} \left[ \sum_{r < t} \alpha_t(r) \cdot \beta_t^*(i) - \sum_{r < t} \alpha_t(r) \beta_r(i) \right]} \] (81)

The interpretation of the reestimation formulas is the following. The formula for \( \overline{\alpha}_i \) is the probability that state \( i \) was the first state, given \( O \). The formula for \( \overline{a}_{i} \) is almost the same as for the usual HMM except it uses the condition that the alpha terms in which a state ends at \( t \), join with the beta terms in which a new state begins at \( t + 1 \). The formula for \( \overline{b}_{i}(k) \) (assuming a discrete density) is the expected number of times that observation \( O_t = v_i \) occurred in state \( i \), normalized by the expected number of times that any observation occurred in state \( i \). Finally, the reestimation formula for \( \overline{p}_{t}(d) \) is the ratio of the expected number of times state \( i \) occurred with duration \( d \) to the expected number of times state \( i \) occurred with any duration.

The importance of incorporating state duration densities is reflected in the observation that, for some problems, the quality of the modeling is significantly improved when explicit state duration densities are used. However, there are drawbacks to the use of the variable duration model discussed in this section. One is the greatly increased computational load associated with using variable durations. It can be seen from the definition and initialization conditions on the forward variable \( \alpha_t(i) \), from (68)–(69), that about \( D \) times the storage and \( D/2 \) times the computation is required. For \( D \) on the order of 25 (as is reasonable for many speech processing problems), computation is increased by a factor of 300. Another problem with the variable duration models is the large number of parameters (\( D \)) associated with each state, which must be estimated, in addition to the usual HMM parameters. Furthermore, for a fixed number of observations \( T \), in the training set, there are, on average, fewer state transitions and much less data to estimate \( p_t(d) \) than would be used in a standard HMM. Thus the reestimation problem is more difficult for variable duration HMMs than for the standard HMM.

One proposal to alleviate some of these problems is to use a parametric state duration density instead of the non-parametric \( p_t(d) \) used above [29], [30]. In particular, proposals include the Gaussian family with

\[ p_t(d) = \mathfrak{R}(d, \mu, \sigma_i^2) \] (82)

with parameters \( \mu_i \) and \( \sigma_i^2 \), or the Gamma family with

\[ p_t(d) = \frac{\eta_t^d \cdot e^{-\eta_t d}}{\Gamma(\nu)} \] (83)

with parameters \( \eta_t \) and \( \nu_t \) and with mean \( \mu_t = \frac{\eta_t}{\nu_t} \) and variance \( \sigma_t^2 = \frac{\eta_t}{\nu_t^2} \). Reestimation formulas for \( \eta_t \) and \( \nu_t \) have been derived and used with good results [19]. Another possibility, which has been used with good success, is to assume a uniform duration distribution (over an appropriate range of durations) and use a path-constrained Viterbi decoding procedure [31].

E. Optimization Criterion—ML, MM, and MD [32], [33]

The basic philosophy of HMMs is that a signal (or observation sequence) can be well modeled if the parameters of an HMM are carefully and correctly chosen. The problem with this philosophy is that it is sometimes inaccurate—either because the signal does not obey the constraints of the HMM, or because it is too difficult to get reliable estimates of all HMM parameters. To alleviate this type of problem, there has been proposed at least two alternatives to the standard maximum likelihood (ML) optimization procedure for estimating HMM parameters.

The first alternative [32] is based on the idea that several HMMs are to be designed and we wish to design them all at the same time in such a way so as to maximize the discrimination power of each model (i.e., each model's ability
to distinguish between observation sequences generated by the correct model and those generated by alternative models. We denote the different HMMs as \( \lambda_1, \lambda_2, \lambda_3, \cdots, \lambda_V \). The standard ML design criterion is to use a separate training sequence of observations \( O' \) to derive model parameters for each model \( \lambda_i \). Thus the standard ML optimization yields

\[
P_i^* = \max_{\lambda_i} P(O' | \lambda_i).
\]  

(84)

The proposed alternative design criterion [31] is the maximum mutual information (MMI) criterion in which the average mutual information \( I \) between the observation sequence \( O' \) and the complete set of models \( \lambda = (\lambda_1, \lambda_2, \cdots, \lambda_V) \) is maximized. One possible way of implementing this is

\[
I^* = \max_{\lambda} \left[ \log P(O' | \lambda) - \log \sum_{\lambda_i} P(O' | \lambda_i) \right]
\]  

(85)
i.e., choose \( \lambda \) so as to separate the correct model \( \lambda_i \) from all other models on the training sequence \( O' \). By summing (85) over all training sequences, one would hope to attain the most separated set of models possible. Thus a possible implementation would be

\[
I^* = \max_{\lambda} \sum_{i=1}^{V} \left[ \log P(O' | \lambda_i) - \log \sum_{i=1}^{V} P(O' | \lambda_i) \right].
\]  

(86)

There are various theoretical reasons why analytical (or reestimation type) solutions to (86) cannot be realized. Thus the only known way of actually solving (86) is via general optimization procedures like the steepest descent methods [32].

The second alternative philosophy is to assume that the signal to be modeled was not necessarily generated by a Markov source, but does obey certain constraints (e.g., positive definite correlation function) [33]. The goal of the design procedure is therefore to choose HMM parameters which minimize the discrimination information (DI) or the cross entropy between the set of valid (i.e., which satisfy the measurements) signal probability densities (call this set \( Q \)) and the set of HMM probability sequences (call this set \( P_\lambda \)), where the DI between \( Q \) and \( P_\lambda \) can generally be written in the form

\[
D(Q \parallel P_\lambda) = \int q(y) \ln \left( q(y) / p(y) \right) dy
\]  

(87)

where \( q \) and \( p \) are the probability density functions corresponding to \( Q \) and \( P_\lambda \). Techniques for minimizing (87) (thereby giving an MDI solution) for the optimum values of \( \lambda = (A, B, \pi) \) are highly nontrivial; however, we use a generalized Baum algorithm as the core of each iteration, and thus are efficiently tailored to hidden Markov modeling [33].

It has been shown that the ML, MMI, and MDI approaches can all be uniformly formulated as MDI approaches. The three approaches differ in either the probability density attributed to the source being modeled, or in the model effectively being used. None of the approaches, however, assumes that the source has the probability distribution of the model.

**F. Comparison of HMMs [34]**

An interesting question associated with HMMs is the following: Given two HMMs, \( \lambda_1 \) and \( \lambda_2 \), what is a reasonable measure of the similarity of the two models? A key point here is the similarity criterion. By way of example, consider the case of two models

\[
\lambda_1 = (A_1, B_1, \pi_1), \quad \lambda_2 = (A_2, B_2, \pi_2)
\]  

with

\[
A_1 = \begin{bmatrix} p & 1-p \\ 1-p & p \end{bmatrix}, \quad B_1 = \begin{bmatrix} q & 1-q \\ 1-q & q \end{bmatrix}, \quad \pi_1 = [1/2 \ 1/2]
\]

and

\[
A_2 = \begin{bmatrix} r & 1-r \\ 1-r & r \end{bmatrix}, \quad B_2 = \begin{bmatrix} s & 1-s \\ 1-s & s \end{bmatrix}, \quad \pi_2 = [1/2 \ 1/2].
\]

For \( \lambda_1 \) to be equivalent to \( \lambda_2 \), in the sense of having the same statistical properties for the observation symbols, i.e., \( p(O_i = v_i | \lambda_1) = p(O_i = v_i | \lambda_2) \), for all \( v_i \), we require

\[
pq + (1-p)(1-q) = rs + (1-r)(1-s)
\]

or, by solving for \( s \), we get

\[
s = \frac{p + q - 2pq}{1 - 2r}.
\]

By choosing (arbitrarily) \( p = 0.6, q = 0.7, r = 0.2 \), we get \( s = 13/30 = 0.433 \). Thus, even when the two models, \( \lambda_1 \) and \( \lambda_2 \), look ostensibly very different (i.e., \( A_1 \) is very different from \( A_2 \) and \( B_1 \) is very different from \( B_2 \)), statistical equivalence of the models can occur.

We can generalize the concept of model distance (dissimilarity) by defining a distance measure \( D(\lambda_1, \lambda_2) \), between two Markov models, \( \lambda_1 \) and \( \lambda_2 \), as

\[
D(\lambda_1, \lambda_2) = \frac{1}{l} \left[ \log P(O'^1 | \lambda_1) - \log P(O'^V | \lambda_2) \right]
\]  

(88)

where \( O'^1 = O_1, O_2, O_3, \cdots, O_T \) is a sequence of observations generated by model \( \lambda_2 \) [34]. Basically (88) is a measure of how well model \( \lambda_1 \) matches observations generated by model \( \lambda_2 \), relative to how well model \( \lambda_1 \) matches observations generated by itself. Several interpretations of (88) exist in terms of cross entropy, or divergence, or discrimination information [34].

One of the problems with the distance measure of (88) is that it is nonsymmetric. Hence a natural expression of this measure is the symmetrized version, namely

\[
D_s(\lambda_1, \lambda_2) = \frac{D(\lambda_1, \lambda_2) + D(\lambda_2, \lambda_1)}{2}.
\]  

(89)

**V. Implementation Issues for HMMs**

The discussion in the previous two sections has primarily dealt with the theory of HMMs and several variations on the form of the model. In this section we deal with several practical implementation issues including scaling, multiple
observation sequences, initial parameter estimates, missing data, and choice of model size and type. For some of these implementation issues we can prescribe exact analytical solutions; for other issues we can only provide some seat-of-the-pants experience gained from working with HMMs over the last several years.

A. Scaling [14]

In order to understand why scaling is required for implementing the reestimation procedure of HMMs, consider the definition of \( \alpha_i(t) \) of (18). It can be seen that \( \alpha_i(t) \) consists of the sum of a large number of terms, each of the form

\[
\prod_{j=1}^{t-1} a_{q_{j} \rightarrow \pi} \prod_{j=t}^{T} b_{q_{j}}(O_{j})
\]

with \( q_{t} = S_{t} \). Since each \( a \) and \( b \) term is less than 1 (generally significantly less than 1), it can be seen that as \( t \) starts to get big (e.g., 10 or more), each term of \( \alpha_i(t) \) starts to head exponentially to zero. For sufficiently large \( t \) (e.g., 100 or more) the dynamic range of the \( \alpha_i(t) \) computation will exceed the precision range of essentially any machine (even in double precision). Hence the only reasonable way of performing the computation is by incorporating a scaling procedure.

The basic scaling procedure which is used is to multiply \( \alpha_i(t) \) by a scaling coefficient that is independent of \( t \) (i.e., it depends only on \( t \), with the goal of keeping the scaled \( \alpha_i(t) \) within the dynamic range of the computer for \( 1 \leq t \leq T \). A similar scaling is done to the \( \beta_i(t) \) coefficients (since these also tend to zero exponentially fast) and then, at the end of the computation, the scaling coefficients are canceled out exactly.

To understand this scaling procedure better, consider the reestimation formula for the state transition coefficients \( a_{ij} \). If we write the reestimation formula (44) directly in terms of the forward and backward variables we get

\[
\alpha_i(t) = \frac{\sum_{j=1}^{T} \alpha_i(j-1) a_{i} b_{j}(O_{j}) \beta_{j}(O_{j})}{\sum_{j=1}^{T} \alpha_i(j-1) a_{i} b_{j}(O_{j}) \beta_{j}(O_{j})}
\]  

(90)

Consider the computation of \( \alpha_i(t) \). For each \( t \), we first compute \( \alpha_i(t) \) according to the induction formula (20), and then we multiply it by a scaling coefficient \( c_t \), where

\[
c_t = \frac{1}{\sum_{j=1}^{N} \alpha_i(j)}
\]  

(91)

Thus, for a fixed \( t \), we first compute

\[
\alpha_i(t) = \sum_{j=1}^{N} \hat{\alpha}_{i-}(j) a_{i} b_{j}(O_{j}).
\]  

(92a)

Then the scaled coefficient set \( \hat{\alpha}_i(t) \) is computed as

\[
\hat{\alpha}_i(t) = \frac{\sum_{j=1}^{N} \hat{\alpha}_{i-}(j) a_{i} b_{j}(O_{j})}{\sum_{j=1}^{N} \hat{\alpha}_{i-}(j) a_{i} b_{j}(O_{j})}
\]  

(92b)

By induction we can write \( \hat{\alpha}_{i-}(j) \) as

\[
\hat{\alpha}_{i-}(j) = \left( \prod_{s=t}^{T} c_s \right) \alpha_{i-}(j).
\]  

(93a)

Thus we can write \( \hat{\alpha}_i(t) \) as

\[
\hat{\alpha}_i(t) = \frac{\sum_{j=1}^{N} \alpha_{i-}(j) a_{i} b_{j}(O_{j})}{\sum_{j=1}^{N} \alpha_{i-}(j) a_{i} b_{j}(O_{j})} \frac{\sum_{j=1}^{N} \alpha_{i-}(j) a_{i} b_{j}(O_{j})}{\sum_{j=1}^{N} \alpha_{i-}(j) a_{i} b_{j}(O_{j})} = \alpha_i(t)
\]  

i.e., each \( \alpha_i(t) \) is effectively scaled by the sum over all states of \( \alpha_i(t) \).

Next we compute the \( \beta_i(t) \) terms from the backward recursion. The only difference here is that we use the same scale factors for each \( t \) for the betas as was used for the alphas. Hence the scaled \( \beta \)'s are of the form

\[
\hat{\beta}_i(t) = c_t \beta_i(t).
\]  

(94)

Since each scale factor effectively restores the magnitude of the \( \alpha \) terms to 1, and since the magnitudes of the \( \alpha \) and \( \beta \) terms are comparable, using the same scaling factors on the \( \beta \)'s as was used on the \( \alpha \)'s is an effective way of keeping the computation within reasonable bounds. Furthermore, in terms of the scaled variables we see that the reestimation equation (90) becomes

\[
\hat{a}_i(t) = \frac{\sum_{j=1}^{T} \hat{\alpha}_i(j) a_{i} b_{j}(O_{j}) \hat{\beta}_i(j)}{\sum_{j=1}^{T} \hat{\alpha}_i(j) a_{i} b_{j}(O_{j}) \hat{\beta}_i(j)}
\]  

(95)

but each \( \hat{\alpha}_i(t) \) can be written as

\[
\hat{\alpha}_i(t) = \left[ \prod_{s=t}^{T} c_s \right] \alpha_i(t) = C_t \alpha_i(t)
\]  

(96)

and each \( \hat{\beta}_i(t) \) can be written as

\[
\hat{\beta}_i(t) = \left[ \prod_{s=t}^{T} c_s \right] \beta_i(t) = D_t \beta_i(t).
\]  

(97)

Thus (95) can be written as

\[
\hat{a}_i(t) = \frac{\sum_{j=1}^{T} C_t \alpha_i(j) a_{i} b_{j}(O_{j}) D_t \beta_i(j)}{\sum_{j=1}^{T} C_t \alpha_i(j) a_{i} b_{j}(O_{j}) D_t \beta_i(j)}
\]  

(98)

Finally the term \( C_t D_{t+1} \) can be seen to be of the form

\[
C_t D_{t+1} = \prod_{s=t}^{T} c_s \prod_{s=t+1}^{T} c_s = \prod_{s=t}^{T} c_s = C_t
\]  

(99)

independent of \( t \). Hence the terms \( C_t D_{t+1} \) cancel out of both the numerator and denominator of (98) and the exact reestimation equation is therefore realized.

It should be obvious that the above scaling procedure applies equally well to reestimation of the \( \pi \) or \( B \) coefficients. It should also be obvious that the scaling procedure of (92) need not be applied at every time instant \( t \), but can be performed whenever desired, or necessary (e.g., to prevent underflow). If scaling is not performed at some instant \( t \), the scaling coefficients \( c_t \) are set to 1 at that time and all the conditions discussed above are then met.

The only real change to the HMM procedure because of scaling is the procedure for computing \( \hat{\pi}(O_t) \). We cannot merely sum up the \( \hat{\alpha}_i(t) \) terms since these are scaled already.
However, we can use the property that
\[
\prod_{t=1}^{T} c_t \sum_{i=1}^{N} \alpha_t(i) = C_t \sum_{i=1}^{N} \alpha_t(i) = 1.
\] (100)
Thus we have
\[
\prod_{t=1}^{T} c_t \cdot P(O|\lambda) = 1
\] (101)
or
\[
P(O|\lambda) = \frac{1}{\prod_{t=1}^{T} c_t}
\] (102)
or
\[
\log P(O|\lambda) = -\sum_{t=1}^{T} \log c_t.
\] (103)
Thus the log of \( P \) can be computed, but not \( P \) since it would be out of the dynamic range of the machine anyway.

Finally we note that when using the Viterbi algorithm to give the maximum likelihood state sequence, no scaling is required if we use logarithms in the following way. (Refer back to (32)–(34).) We define
\[
\phi_t(i) = \max_{q_t, q_{t+1}, \ldots, q_T} \{ \log P(q_t, q_{t+1}, \ldots, q_T, O_t, O_{t+1}, \ldots, O_T|\lambda) \}
\] (104)
and initially set
\[
\phi_1(i) = \log (\pi_i) + \log [b_i(O_1)]
\] (105a)
with the recursion step
\[
\phi_t(i) = \max_{1 \leq s \leq N} [\phi_{t-1}(s) + \log a_{s,i} + \log [b_i(O_t)]]
\] (105b)
and the termination step
\[
\log P^* = \max_{1 \leq i \leq N} [\phi_T(i)].
\] (105c)
Again we arrive at \( \log P^* \) rather than \( P^* \), but with significantly less computation and with no numerical problems. (The reader should note that the terms \( \log a_{s,i} \) of (105b) can be precomputed and therefore do not cost anything in the computation. Furthermore, the terms \( \log [b_i(O_t)] \) can be precomputed when a finite observation symbol analysis (e.g., a codebook of observation sequences) is used.

B. Multiple Observation Sequences [14]

In Section IV we discussed a form of HMM called the left-right or Bakis model in which the state proceeds from state 1 at \( t = 1 \) to state \( N \) at \( t = T \) in a sequential manner (recall the model of Fig. 7(b)). We have already discussed how a left-right model imposes constraints on the state transition matrix, and the initial state probabilities (45)–(48). However, the major problem with left-right models is that one cannot use a single observation sequence to train the model (i.e., for reestimation of model parameters). This is because the transient nature of the states within the model only allow a small number of observations for any state (until a transition is made to a successor state). Hence, in order to have sufficient data to make reliable estimates of all model parameters, one has to use multiple observation sequences.

The modification of the reestimation procedure is straightforward and goes as follows. We denote the set of \( K \) observation sequences as
\[
O = \{O^{(1)}, O^{(2)}, \ldots, O^{(K)}\}
\] (106)
where \( O^{(k)} = \{O^{(k)}_1, O^{(k)}_2, \ldots, O^{(k)}_T\} \) is the \( k \)th observation sequence. We assume each observation sequence is independent of every other observation sequence, and our goal is to adjust the parameters of the model \( \lambda \) to maximize
\[
P(O|\lambda) = \prod_{k=1}^{K} P(O^{(k)}|\lambda)
\] (107)
\[
= \prod_{k=1}^{K} P_k.
\] (108)
Since the reestimation formulas are based on frequencies of occurrence of various events, the reestimation formulas for multiple observation sequences are modified by adding together the individual frequencies of occurrence for each sequence. Thus the modified reestimation formulas for \( \bar{a}_i \) and \( \bar{b}_j(i) \) are
\[
\bar{a}_i = \frac{1}{K} \sum_{k=1}^{K} \sum_{s=1}^{T-1} \alpha_t^k(i) a_{s,i} b_s(O^{(k)}_t) \beta_t^k(i), \quad j(i)
\] (109)
and
\[
\bar{b}_j(i) = \frac{1}{K} \sum_{k=1}^{K} \sum_{s=1}^{T-1} \alpha_t^k(i) b_t^k(i)
\] (110)
and \( \pi_i \) is not reestimated since \( \pi_1 = 1, \pi_i = 0, i \neq 1 \).

The proper scaling of (109)–(110) is now straightforward since each observation sequence has its own scaling factor. The key idea is to remove the scaling factor from each term before summing. This can be accomplished by writing the reestimation equations in terms of the scaled variables, i.e.,
\[
\bar{a}_i = \frac{1}{K} \sum_{k=1}^{K} \sum_{s=1}^{T-1} \alpha_t^k(i) a_{s,i} b_s(O^{(k)}_t) \beta_t^k(i), \quad j(i)
\] (111)
\[
\sum_{k=1}^{K} \sum_{s=1}^{T-1} \alpha_t^k(i) b_t^k(i).
\]
In this manner, for each sequence \( O^{(k)} \), the same scale factors will appear in each term of the sum over \( t \) as appears in the \( P_k \) term, and hence will cancel exactly. Thus using the scaled values of the alphas and betas results in an unscaled \( \bar{a}_i \). A similar result is obtained for the \( \bar{b}_j(i) \) term.

C. Initial Estimates of HMM Parameters

In theory, the reestimation equations should give values of the HMM parameters which correspond to a local maximum of the likelihood function. A key question is therefore how do we choose initial estimates of the HMM parameters so that the local maximum is the global maximum of the likelihood function.

Basically there is no simple or straightforward answer to the above question. Instead, experience has shown that either random (subject to the stochastic and the nonzero value constraints) or uniform initial estimates of the \( \pi \) and
A parameters is adequate for giving useful reestimates of these parameters in almost all cases. However, for the $B$ parameters, experience has shown that good initial estimates are helpful in the discrete symbol case, and are essential (when dealing with multiple mixtures) in the continuous distribution case [35]. Such initial estimates can be obtained in a number of ways, including manual segmentation of the observation sequence(s) into states with averaging of observations within states, maximum likelihood segmentation of observations with averaging, and segmental k-means segmentation with clustering, etc. We discuss such segmentation techniques later in this paper.

D. Effects of Insufficient Training Data [36]

Another problem associated with training HMM parameters via reestimation methods is that the observation sequence used for training is, of necessity, finite. Thus there is often an insufficient number of occurrences of different model events (e.g., symbol occurrences within states) to give good estimates of the model parameters. One solution to this problem is to increase the size of the training observation set. Often this is impractical. A second possible solution is to reduce the size of the model (e.g., number of states, number of symbols per state, etc). Although this is always possible, often there are physical reasons why a given model is used and therefore the model size cannot be changed.

A third possible solution is to interoperate one set of parameter estimates with another set of parameter estimates from a model for which an adequate amount of training data exists [36]. The idea is to simultaneously design both the desired model as well as a smaller model for which the amount of training data is adequate to give good parameter estimates, and then to interoperate the parameter estimates from the two models. The way in which the smaller model is chosen is by tying one or more sets of parameters of the initial model to create the smaller model. Thus if we have estimates for the parameters for the model $\lambda = (A, B, \pi)$, as well as for the reduced size model $\lambda' = (A', B', \pi')$, then the interpolated model, $\hat{\lambda} = (A, B, \pi)$, is obtained as

$$\hat{\lambda} = \lambda + (1 - \epsilon)\lambda'$$  \hspace{1cm} (112)

where $\epsilon$ represents the weighting of the parameters of the full model, and $(1 - \epsilon)$ represents the weighting of the parameters of the reduced model. A key issue is the determination of the optimal value of $\epsilon$, which is clearly a function of the amount of training data. (As the amount of training data gets large, we expect $\epsilon$ to tend to 1.0; similarly for small amounts of training data we expect $\epsilon$ to tend to 0.0.)

The solution to the determination of an optimal value for $\epsilon$ was provided by Jelinek and Mercer [36] who showed how the optimal value for $\epsilon$ could be estimated using the forward-backward algorithm by interpreting (112) as an expanded HMM of the type shown in Fig. 10. For this expanded model the parameter $\epsilon$ is the probability of a state transition from the (neutral) state $s$ to the model $\lambda$; similarly $(1 - \epsilon)$ is the probability of a state transition from $s$ to the model $\lambda'$. Between each of the models, $\lambda$ and $\lambda'$, and $s$, there is a null transition. Using the model of Fig. 9, the value of $\epsilon$ can be estimated from the training data in the standard manner. A key point is to segment the training data $T$ into two disjoint sets, i.e., $T = T_1 \cup T_2$. Training set $T_1$ is first used to train models $\lambda$ and $\lambda'$ (i.e., to give estimates of $A$, $B$, $\pi$ and $A'$, $B'$, $\pi'$). Training set $T_2$ is then used to give an estimate of $\epsilon$, assuming the models $\lambda$ and $\lambda'$ are fixed. A modified version of this training procedure, called the method of deleted interpolation [36], iterates the above procedure through multiple partitions of the training set. For example one might consider a partition of the training set such that $T_1$ is 90 percent of $T$ and $T_2$ is the remaining 10 percent of $T$. There are a large number of ways in which such a partitioning can be accomplished but one particularly simple one is to cycle $T_2$ through the data, i.e., the first partition uses the last 10 percent of the data as $T_2$, the second partition uses the next-to-last 10 percent of the data as $T_2$, etc.

The technique of deleted interpolation has been successfully applied to a number of problems in speech recognition including the estimation of trigram word probabilities for language models [13], and the estimation of HMM output probabilities for trigram phone models [37], [38].

Another way of handling the effects of insufficient training data is to add extra constraints to the model parameters to insure that no model parameter estimate falls below a specified level. Thus, for example, we might specify the constraint, for a discrete symbol model, that

$$b_j(k) \geq \delta$$ \hspace{1cm} (113a)

or, for a continuous distribution model, that

$$U_{jk}(r, \theta) \geq \delta.$$ \hspace{1cm} (113b)

The constraints can be applied as a postprocessor to the reestimation equations such that if a constraint is violated, the relevant parameter is manually corrected, and all remaining parameters are rescaled so that the densities obey the required stochastic constraints. Such post-processor techniques have been applied to several problems in speech processing with good success [39]. It can be seen from (112) that this procedure is essentially equivalent to a simple form of deleted interpolation in which the model $\lambda'$ is a uniform distribution model, and the interpolation value $\epsilon$ is chosen as the fixed constant $(1 - \delta)$.

E. Choice of Model

The remaining issue in implementing HMMs is the choice of type of model (ergodic or left-right or some other form), choice of model size (number of states), and choice of observation symbols (discrete or continuous, single or multi-mixture, choice of observation parameters). Unfortunately, there is no simple, theoretically correct, way of making such choices. These choices must be made depending on the signal being modeled. With these comments we
end our discussion of the theoretical aspects of hidden Markov models, and proceed to a discussion of how such models have been applied to selected problems in speech recognition.

VI. Implementation of Speech Recognizers Using HMMs

The purpose of this, and the following sections, is to illustrate how the ideas of HMMs, as discussed in the first five sections of this paper, have been applied to selected problems in speech recognition. As such, we will not strive to be as thorough or as complete in our descriptions as to what was done as we were in describing the theory of HMMs. The interested reader should read the material in [6], [10], [12], [13], [39]–[46] for more complete descriptions of individual systems. Our main goal here is to show how specific aspects of HMM theory get applied, not to make the reader an expert in speech recognition technology.

A. Overall Recognition System

Fig. 11 shows a block diagram of a pattern recognition approach to continuous speech recognition system. The key signal processing steps include the following:

1) Feature Analysis: A spectral and/or temporal analysis of the speech signal is performed to give observation vectors which can be used to train the HMMs which characterize various speech sounds. A detailed discussion of one type of feature analysis is given later in this section.

2) Unit Matching System: First a choice of speech recognition unit must be made. Possibilities include linguistically based sub-word units such as phones (or phone-like units), diphones, demisyllables, and syllables [38], as well as derivative units such as phonemes, phonones, and acoustic units [13]. Other possibilities include whole word units, and even units which correspond to a group of 2 or more words (e.g., and, in, the, of a, etc). Generally, the less complex the unit (e.g., phones), the fewer of them there are in the language, and the more complicated (variable) their structure in continuous speech. For large vocabulary speech recognition (involving 1000 or more words), the use of sub-word speech units is almost mandatory as it would be quite difficult to record an adequate training set for designing HMMs for units of the size of words or larger. However, for specialized applications (e.g., small vocabulary, constrained task), it is both reasonable and practical to consider the word as a basic speech unit. We will consider such systems exclusively in this and the following section. Independent of the unit chosen for recognition, an inventory of such units must be obtained via training. Typically each such unit is characterized by some type of HMM whose parameters are estimated from a training set of speech data. The unit matching system provides the likelihoods of a match of all sequences of speech recognition units to the unknown input speech. Techniques for providing such match scores, and in particular determining the best match score (subject to lexical and syntactic constraints of the system) include the stack decoding procedure [7], various forms of frame synchronous path decoding [37], and a lexical access scoring procedure [46].

3) Lexical Decoding: This process places constraints on the unit matching system so that the paths investigated are those corresponding to sequences of speech units which are in a word dictionary (a lexicon). This procedure implies that the speech recognition word vocabulary must be specified in terms of the basic units chosen for recognition. Such a specification can be deterministic (e.g., one or more finite state networks for each word in the vocabulary) or statistical (e.g., probabilities attached to the arcs in the finite state representation of words). In the case where the chosen units are words (or word combinations), the lexical decoding step is essentially eliminated and the structure of the recognizer is greatly simplified.

4) Syntactic Analysis: This process, much like lexical decoding, places further constraints on the unit matching system so that the paths investigated are those corresponding to speech units which comprise words (lexical decoding) and for which the words are in a proper sequence as specified by a word grammar. Such a word grammar can again be represented by a deterministic finite state network (in which all word combinations which are accepted by the grammar are enumerated), or by a statistical grammar (e.g., a trigram word model in which probabilities of sequences of 3 words in a specified order are given). For some command and control tasks, only a single word from a finite set of equiprobable is required to be recognized and therefore the grammar is either trivial or unnecessary. Such tasks are often referred to as isolated word speech recognition tasks. For other applications (e.g., digit sequences) very simple grammars are often adequate (e.g., any digit can be spoken and followed by any other digit). Finally there are tasks for which the grammar is a dominant factor and, although it adds a great deal of constraint to the recognition process, it greatly improves recognition performance by the resulting restrictions on the sequence of speech units which are valid recognition candidates.

5) Semantic Analysis: This process, again like the steps of syntactic analysis and lexical decoding, adds further constraints to the set of recognition search paths. One way in which semantic constraints are utilized is via a dynamic model of the state of the recognizer. Depending on the recognizer state certain syntactically correct input strings are eliminated from consideration. This again serves to make the recognition task easier and leads to higher performance of the system.

![Block diagram of a continuous speech recognizer.](image-url)
There is one additional factor that has a significant effort on the implementation of a speech recognizer and that is the problem of separating background silence from the input speech. There are at least three reasonable ways of accomplishing this task:

1) Explicitly detecting the presence of speech via techniques which discriminate background from speech on the basis of signal energy and signal durations. Such methods have been used for template-based approaches because of their inherent simplicity and their success in low to moderate noise backgrounds [48].

2) Build a model of the background silence, e.g., a statistical model, and represent the incoming signal as an arbitrary sequence of speech and background, i.e.,

\[
\text{signal} = \text{(silence)} - \text{speech} - \text{(silence)}
\]

where the silence part of the signal is optional in that it may not be present before or after the speech [49].

3) Extend the speech unit models so that background silence is included (optionally) within the first and/or last state of the model, and therefore silence inherently gets included within all speech unit models.

All three of these techniques have been utilized in speech recognition systems.

Instead of discussing the general continuous speech recognition system further, we now present specialized applications to illustrate how HMM technology can be utilized. First we present a system where the basic speech unit is the word, where the task is to recognize a single spoken word, and where there is no task syntax or semantics to constrain the choice of words. This task is generally referred to as isolated word recognition. Next we discuss a slightly more complicated task in which the basic speech unit is still the word, but where the task is to recognize a continuous utterance consisting of words from the vocabulary. Included in such a task is the problem of recognizing a spoken string of digits. We again consider the case where there is no task syntax or semantics to constrain the choice of words, i.e., any digit can follow any other digit. Recognition tasks of this type have been referred to as connected word recognizers because the continuous speech is recognized as a concatenated sequence of word models. This is technically a misnomer because it is truly a continuous speech recognition problem. However, the terminology has become established and we continue its use.

**B. Isolated Word Recognition**

As our first example, consider using HMMs to build an isolated word recognizer. Assume we have a vocabulary of \( V \) words to be recognized and that each word is to be modeled by a distinct HMM. Further assume that for each word in the vocabulary we have a training set of \( K \) occurrences of each spoken word (spoken by \( T \) or more talkers) where each occurrence of the word constitutes an observation sequence, where the observations are some appropriate representation of the (spectral and/or temporal) characteristics of the word. (We will return to the question of what specific representation is used later in this section.) In order to do isolated word speech recognition, we must perform the following:

1) For each word \( v \) in the vocabulary, we must build a HMM \( \lambda^v \), i.e., we must estimate the model parameters \((A, B, \pi)\) that optimize the likelihood of the training set observation vectors for the \( v \)th word.

2) For each unknown word which is to be recognized, the processing of Fig. 12 must be carried out, namely measurement of the observation sequence \( O = \{O_1, O_2, \ldots, O_T\} \), via a feature analysis of the speech corresponding to the word; followed by calculation of model likelihoods for all possible models, \( P(O | \lambda^v) \), \( 1 \leq v \leq V \); followed by selection of the word whose model likelihood is highest, i.e.,

\[

\begin{align*}
\nu^* &= \underset{1 \leq v \leq V}{\text{argmax}} \ P(O | \lambda^v). \\
\end{align*}
\]

(114)

The probability computation step is generally performed using the Viterbi algorithm (i.e., the maximum likelihood path is used) and requires on the order of \( V \cdot N^2 \cdot T \) computations. For modest vocabulary sizes, e.g., \( V = 100 \) words, with an \( N = 5 \) state model, and \( T = 40 \) observations for the

![Fig. 12. Block diagram of an isolated word HMM recognizer.](image)
unknown word, a total of $10^6$ computations is required for recognition (where each computation is a multiply, and add, and a calculation of observation density, $b(o)$). Clearly this amount of computation is modest as compared to the capabilities of most modern signal processor chips.

C. LPC Feature Analysis [51]–[54]

One way to obtain observation vectors $O$ from speech samples $s$ is to perform a front end spectral analysis. (We assume that we are processing only the speech samples corresponding to the spoken word—i.e., all background before and after the spoken word has been eliminated by an appropriate word detection algorithm.) The type of spectral analysis that is often used (and the one we will describe here) is called linear predictive coding (LPC), and a block diagram of the steps that are carried out is given in Fig. 13. The overall system is a block processing model in which a frame of $N_A$ samples is processed and a vector of features $Q_A$ is computed. The steps in the processing are as follows:

1) **Preemphasis**: The digitized (at a 6.67 kHz rate for the examples to be discussed here) speech signal is processed by a first-order digital network in order to spectrally flatten the signal.

2) **Blocking into Frames**: Sections of $N_A$ consecutive speech samples (we use $N_A = 300$ corresponding to 45 ms of signal) are used as a single frame. Consecutive frames are spaced $M_A$ samples apart (we use $M_A = 100$ corresponding to 15-ms frame spacing, or 30-ms frame overlap).

3) **Frame Windowing**: Each frame is multiplied by an $N_A$-sample window (we use a Hamming window $w(n)$) so as to minimize the adverse effects of chopping an $N_A$-sample section out of the running speech signal.

4) **Autocorrelation Analysis**: Each windowed set of speech samples is autocorrelated to give a set of $(p + 1)$ coefficients, where $p$ is the order of the desired LPC analysis (we use $p = 8$).

5) **LPC/Cepstral Analysis**: For each frame, a vector of LPC coefficients is computed from the autocorrelation vector using a Levinson or a Durbin recursion method. An LPC derived cepstral vector is then computed up to the $Q$th component, where $Q > p$ and $Q = 12$ in the results to be described later in this section.

6) **Cepstral Weighting**: The $Q$-coefficient cepstral vector $C_q(m)$ at time frame $t$ is weighted by a window $W_q(m)$ of the form [55], [56]

$$W_q(m) = 1 + \frac{Q}{2} \sin \left( \frac{\pi m}{Q} \right), \quad 1 \leq m \leq Q \quad (115)$$

to give

$$C_q(m) = c_q(m) \cdot W_q(m), \quad (116)$$

7) **Delta Cepstrum**: The time derivative of the sequence of weighted cepstral vectors is approximated by a first-order orthogonal polynomial over a finite length window of $(2K + 1)$ frames, centered around the current vector [57], [58]. $(K = 2$ in the results to be presented; hence a 5 frame window is used for the computation of the derivative.) The cepstral derivative (i.e., the delta cepstrum vector) is computed as

$$\Delta C_q(m) = \sum_{k=-k}^{k} \left[ \sum_{n=0}^{\infty} k C_{q-k}(m) \right] \cdot G, \quad 1 \leq m \leq Q \quad (117)$$

where $G$ is a gain term chosen to make the variances of $C_q(m)$ and $\Delta C_q(m)$ equal. (A value of $G$ of 0.375 was used.)

The observation vector $Q_q$ used for recognition and training is the concatenation of the weighted cepstral vector, and the corresponding weighted delta cepstrum vector, i.e.,

$$Q_q = \{C_q(m), \Delta C_q(m)\} \quad (118)$$

and consists of 24 coefficients per vector.

D. Vector Quantization [18], [39]

For the case in which we wish to use an HMM with a discrete observation symbol density, rather than the continuous vectors above, a vector quantizer (VQ) is required to map each continuous observation vector into a discrete codebook index. Once the codebook of vectors has been obtained, the mapping between continuous vectors and

---

Fig. 13. Block diagram of the computations required in the front end feature analysis of the HMM recognizer.
codebook indices becomes a simple nearest neighbor computation, i.e., the continuous vector is assigned the index of the nearest (in a spectral distance sense) codebook vector. Thus the major issue in VQ is the design of an appropriate codebook for quantization.

Fortunately a great deal of work has gone into devising an excellent iterative procedure for designing codebooks based on having a representative training sequence of vectors [18]. The procedure basically partitions the training vectors into M disjoint sets (where M is the size of the codebook), represents each such set by a single vector \( \mathbf{v}_m \) \( (1 \leq m \leq M) \), which is generally the centroid of the vectors in the training set assigned to the \( m \)th region, and then iteratively optimizes the partition and the codebook (i.e., the centroids of each partition). Associated with VQ is a distortion penalty since we are representing an entire region of the vector space by a single vector. Clearly it is advantageous to keep the distortion penalty as small as possible. However, this implies a large size codebook, and that leads to problems in implementing HMMs with a large number of parameters. Fig. 14 illustrates the tradeoff of quantization from 2 to 10 states would be appropriate. The other idea is to let the number of states correspond roughly to the average number of observations in a spoken version of the word, the so-called Bakis model [11]. In this manner each state corresponds to an observation interval—i.e., about 15 ms for the analysis we use. In the results to be described later in this section, we use the former approach. Furthermore we restrict each word model to have the same number of states; this implies that the models will work best when they represent words with the same number of sounds.

To illustrate the effect of varying the number of states in a word model, Fig. 15 shows a plot of average word error rate versus \( N \), for the case of recognition of isolated digits (i.e., a 10-word vocabulary). It can be seen that the error rate is somewhat insensitive to \( N \), achieving a local minimum at \( N = 6 \); however, differences in error rate for values of \( N \) close to 6 are small.

The next issue is the choice of observation vector and the way it is represented. As discussed in Sections VI-C and VI-D, we have considered LPC derived weighted cepstral coefficients and weighted cepstral derivatives or (for auto-regressive HMMs) the autocorrelation of the LPC coefficients as the observation vectors for continuous models; for discrete symbol models we use a codebook to generate the discrete symbols. For the continuous models we use as many as \( M = 9 \) mixtures per state; for the discrete symbol models we use codebooks with as many as \( M = 256 \) codewords. Also, for the continuous models, we have found that it is preferable to use diagonal covariance matrices with several mixtures, rather than fewer mixtures with full covariance matrices. The reason for this is simple, namely the difficulty in performing reliable reestimation of the off-diagonal components of the covariance matrix from the necessarily limited training data. To illustrate the need for using mixture densities for modeling LPC observation vectors (i.e., eighth-order cepstral vectors with log energy appended as the ninth vector component), Fig. 16 shows a comparison of marginal distributions \( b_j(O|\theta_{0\ldots n\ldots}) \) against a histogram of the actual observations within a state (as determined by a maximum likelihood segmentation of all the training observations into states). The observation vectors are ninth order, and the model density uses \( M = 5 \) mixtures. The covariance matrices are constrained to be diagonal for each individual mixture. The results of Fig. 16 are for the first model state of the word "zero." The need for values of \( M > 1 \) is clearly seen in the histogram of the.
first parameter (the first cepstral component) which is inherently multimodal; similarly the second, fourth, and eight cepstral parameters show the need for more than a single Gaussian component to provide good fits to the empirical data. Many of the other parameters appear to be well fitted by a single Gaussian; in some cases, however, even $M = 5$ mixtures do not provide a sufficiently good fit.

Another experimentally verified fact about the HMM is that it is important to limit some of the parameter estimates in order to prevent them from becoming too small. For example, for the discrete symbol models, the constraint that $b_j(k)$ be greater than or equal to some minimum value $\epsilon$ is necessary to insure that even when the $k$th symbol never occurred in some state $j$ in the training observation set, there is always a finite probability of its occurrence when scoring an unknown observation set. To illustrate this point, Fig. 17 shows a curve of average word error rate versus the parameter $\epsilon$ (on a log scale) for a standard word recognition experiment. It can be seen that over a very broad range ($10^{-10} \leq \epsilon \leq 10^{-4}$) the average error rate remains at about a constant value; however, when $\epsilon$ is set to 0 (i.e., $10^{-\infty}$), then the error rate increases sharply. Similarly, for continuous densities it is important to constrain the mixture gains $c_m$ as well as the diagonal covariance coefficients $U_{\mu \mu}(r, r)$ to be greater than or equal to some minimum values (we use $10^{-4}$ in all cases).

F. Segmental $k$-Means Segmentation into States [42]

We stated earlier that good initial estimates of the parameters of the $b_j(O)$ densities were essential for rapid and proper convergence of the reestimation formulas. Hence a procedure for providing good initial estimates of these parameters was devised and is shown in Fig. 18. The training procedure is a variant on the well-known $K$-means iterative procedure for clustering data.

We assume we have a training set of observations (the same as is required for parameter reestimation), and an initial estimate of all model parameters. However, unlike the one required for reestimation, the initial model estimate can be chosen randomly, or on the basis of any available model which is appropriate to the data.

Following model initialization, the set of training observation sequences is segmented into states, based on the current model $\lambda$. This segmentation is achieved by finding the optimum state sequence, via the Viterbi algorithm, and then backtracking along the optimal path. This procedure is illustrated in Fig. 19 which shows a log-energy plot, an accumulated log-likelihood plot, and a state segmentation for one occurrence of the word "six." It can be seen in Fig. 19 that the states correspond roughly to the sounds in the spoken word "six."

The result of segmenting each of the training sequences is, for each of the $N$ states, a maximum likelihood estimate of the set of the observations that occur within each state $S_j$, according to the current model. In the case where we are using discrete symbol densities, each of the observation vectors within a state is coded using the $M$-codeword codebook, and the updated estimate of the $b_j(k)$ parameters is

$$ b_j(k) = \text{number of vectors with codebook index } k \text{ in state } j \text{ divided by the number of vectors in state } j. $$

The current or initial model could be one created from another set of talkers, or it could be one created from a uniform segmentation of each word into states.
In the case where we are using continuous observation densities, a segmental $K$-means procedure is used to cluster the observation vectors within each state $S_j$ into a set of $M$ clusters (using a Euclidean distortion measure), where each cluster represents one of the $M$ mixtures of the $b_i(O)$ density. From the clustering, an updated set of model parameters is derived as follows:

\[ c_{jm} = \text{number of vectors classified in cluster } m \text{ of state } j \]
\[ \hat{\mu}_{jm} = \text{sample mean of the vectors classified in cluster } m \text{ of state } j \]
\[ \hat{\Sigma}_{jm} = \text{sample covariance matrix of the vectors classified in cluster } m \text{ of state } j. \]

Based on this state segmentation, updated estimates of the $a_{ij}$ coefficients can be obtained by counting the number of transitions from state $i$ to $j$ and dividing it by the number of transitions from state $i$ to any state (including itself).

An updated model $\hat{\lambda}$ is obtained from the new model parameters and the formal reestimation procedure is used to reestimate all model parameters. The resulting model is then compared to the previous model (by computing a distance score that reflects the statistical similarity of the HMMs). If the model distance score exceeds a threshold, then the old model $\lambda$ is replaced by the new (reestimated) model $\hat{\lambda}$, and the overall training loop is repeated. If the model distance score falls below the threshold, then model convergence is assumed and the final model parameters are saved.

G. Incorporation of State Duration into the HMM

In Section IV-C we discussed the theoretically correct method of incorporating state duration information into the mechanics of the HMM. We also showed that the cost of including duration density was rather high; namely a $D^2$-fold increase in computation and a $D$-fold increase in storage. Using a value of $D = 25$ (as is required for word recognition), the cost of the increased computation tended to make the techniques not worth using. Thus the following alternative procedure was formulated for incorporating state duration information into the HMM.

For this alternative procedure, the state duration probability $p_j(d)$ was measured directly from the segmented training sequences used in the segmental $K$-means procedure of the previous section. Hence the estimates of $p_j(d)$ are strictly heuristic ones. A typical set of histograms of $p_j(d)$ for a 5-state model of the word “six” is shown in Fig. 20. (In this figure the histograms are plotted versus normalized duration ($d/T$), rather than absolute duration $d$.) It can be seen from Fig. 20 that the first two states account for the initial /s/ in “six”; the third state accounts for the transition to the vowel /i/; the fourth state accounts for the vowel; and the fifth state accounts for the stop and the final /s/ sound.

The way in which the heuristic duration densities were used in the recognizer was as follows. First the normal Viterbi algorithm is used to give the best segmentation of the observation sequence of the unknown word into states via a backtracking procedure. The duration of each state is then measured from the state segmentation. A postprocessor then increments the log-likelihood score of the Viterbi algorithm, by the quantity

\[ \log \hat{p}(q, O | \lambda) = \log p(q, O | \lambda) + \alpha_q \sum_{j=1}^{N} \log [p_j(d)] \]  

where $\alpha_q$ is a scaling multiplier on the state duration scores, and $d_j$ is the duration of state $j$ along the optimal path as determined by the Viterbi algorithm. The incremental cost of the postprocessor for duration is essentially negligible, and experience has shown that recognition performance...
is essentially as good as that obtained using the theoretically correct duration model.

H. HMM Performance on Isolated Word Recognition

We conclude this section on isolated word recognition using HMMs by giving a set of performance results (in terms of average word error rate) on the task of recognizing isolated digits in a speaker independent manner. For this task, a training set consisting of 100 occurrences of each digit by 100 talkers (i.e., a single occurrence of each digit per talker) was used. Half the talkers were male; half female. For testing the algorithm, we used the initial training set, as well as three other independent test sets with the following characteristics:

| TS2: | the same 100 talkers as were used in the training; 100 occurrences of each digit |
| TS3: | a new set of 100 talkers (50 male, 50 female); 100 occurrences of each digit |
| TS4: | another new set of 100 talkers (50 male, 50 female); 100 occurrences of each digit |

The results of the recognition tests are given in Table 1. The recognizers are the following:

- **LPC/DTW**: Conventional template-based recognizer using dynamic time warping (DTW) alignment
- **LPC/DTW/VQ**: Conventional recognizer with vector quantization of the feature vectors ($M = 64$)
- **HMM/VQ**: HMM recognizer with $M = 64$ codebook
- **HMM/CD**: HMM recognizer using continuous density model with $M = 5$ mixtures per state
- **HMM/AR**: HMM recognizer using autoregressive observation density

| Table 1 Average Digit Error Rates for Several Recognizers and Evaluation Sets |
|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|
| Recognizer Type             | Evaluation Set              | Original Training | TS2 | TS3 | TS4 |
| LPC/DTW                     |                             | 0.1              | 0.2 | 2.0 | 1.1 |
| LPC/DTW/VQ                  |                             | -                | 3.5 | -   | -   |
| HMM/VQ                      |                             | 0.7              | 0.2 | 1.3 | 1.8 |
| HMM/CD                      |                             | 0.3              | 1.8 | 3.4 | 4.1 |

It can be seen that, when using a VQ, the performance of the isolated word recognizer degrades in both the conventional and HMM modes. It can also be seen that the performances of the conventional template-based recognizer, and the HMM recognizer with a continuous density model are comparable. Finally Table 1 shows that the autoregressive density HMM gives poorer performance than the standard mixture density model.

VII. CONNECTED WORD RECOGNITION USING HMMs [59]-[63]

A somewhat more complicated problem of speech recognition, to which HMMs have been successfully applied, is the problem of connected word recognition. The basic premise of connected word recognition is that the recognition is based on individual word models (as opposed to models of speech units smaller than words). The recognition problem (once the appropriate word models have been derived) is to find the optimum sequence (concatenation) of word models that best matches (in a maximum likelihood sense) an unknown connected word string. In this section we discuss one method (called the level building approach) for solving for such optimum sequences of word models. An alternative method for obtaining the optimum sequence of words is a frame (time) synchronous Viterbi search [31]. There are several practical advantages of the frame synchronous search (e.g., ease of real-time hardware implementation, ease of path pruning, etc.) but these do not affect the optimality of the two methods. For convenience, we restrict our discussion to the recognition of strings of connected digits.

A. Connected Digit Recognition from Word HMMs Using Level Building

A block diagram of the overall level building connected digit recognizer is given in Fig. 21. There are essentially three steps in the recognition process:

1) **Spectral Analysis**: The speech signal $s(n)$ is converted to either a set of LPC vectors or a set of cepstral and delta cepstral vectors. This defines the observation sequence $O$ of the unknown connected digit string.

2) **Level Building**: The sequence of spectral vectors (the observations) of the unknown connected digit string is matched against the single word HMMs using a Viterbi scoring algorithm. The output of this process is a set of candidate digit strings, generally of different lengths (i.e., different number of digits per string), ordered by log probability scores.

3) **Postprocessor**: The candidate digit strings are subjected to further validity tests (e.g., duration), to eliminate unreasonable (unlikely) candidates. The postprocessor chooses the most likely digit string from the remaining (valid) candidate strings.

Individual digits are each characterized by an HMM of the type shown in Fig. 22. (Transitions between words are handled by a switch mode from the last state of one word model, to the first state of another word model, in the level building implementation.) The parameters of the HMMs used for characterizing digits are the following:

1) $N = 5$ or 8 states for digit models trained from observations of a single talker, and $N = 8$ or 10 states, for...
digit models trained from observations of more than
a single talker.

2) Continuous observation mixture densities with \( M = 3 \) or \( 5 \) mixtures per state for single talker models and \( M = 9 \) mixtures per state for multiple talker models.

3) Energy probability \( p_i(e) \) where \( 
\begin{align*}
  p_i(e) &= \frac{1}{Z_i} e^{-\beta_i e} \\
  \beta_i &= \text{a scaling parameter}
\end{align*}
\)
which is the dynamically normalized log energy of the frame of speech used to give observation vector \( O_t \), and \( p_i(e) \) is a discrete density of log energy values in state \( j \). The density is derived empirically from the training data.

4) State duration density \( p_i(D) \), \( 1 \leq d \leq D = 25 \).

In addition to the observation density, log energy probability, and state duration density, each word HMM \( \lambda \) is also characterized by an overall word duration density \( p_i(D) \) of the form

\[
p_i(D) = \pi_i(\tilde{D}_v, \sigma_v^2)
\]

where \( \tilde{D}_v \) is the average duration for word \( v \), \( \sigma_v^2 \) is the variance in duration for word \( v \), and \( \pi_i \) is the normal density.

**B. Level Building on HMMs**

The way in which level building is used on HMMs is illustrated in Fig. 23. If we denote the set of \( V \) word HMMs as \( \lambda \), \( 1 \leq v \leq V \), then to find the optimum sequence of HMMs that match \( O \) (i.e., maximize the likelihood), a sequence of Viterbi matches is performed. For each HMM \( \lambda_v \) and at each level \( \ell \), we do a Viterbi match against \( O \), starting at frame (observation interval) 1 on level 1, and retain for each possible frame \( t \) the following:

1) \( P(t), 1 \leq t \leq T \), the accumulated log probability to frame \( t \), at level \( \ell \), for reference model \( \lambda_v \), along the best path.
2) \( F(t), 1 \leq t \leq T \), a backpointer indicating where the path started at the beginning of the level.

To compute \( P(t) \), we need a local measure for the probability that observation \( O_t \) with log energy \( e_t \), occurred in state \( j \) of model \( \lambda_v \). We use, as the observation density, the function

\[
\hat{\beta}_i(O_t) = b_i(O_t) \cdot [p_i(e)]^\gamma_i \cdot K_i
\]

where \( \gamma_i \) (set to 0.375) is a log energy scaling coefficient and \( K_i \) is a normalization constant. The state transition coefficients enter the calculation of \( P(t) \) via the dynamic programming optimization in determining the Viterbi path.

At the end of each level \( \ell \) (where the level corresponds to word position within the string), a maximization over \( v \) is performed to get the best model at each frame \( t \) as follows:

\[
P_i(t) = \max_{1 \leq v \leq V} p_i(t), 1 \leq t \leq T
\]

\[
W(t) = \arg\max_{1 \leq v \leq V} P_i(t), 1 \leq t \leq T
\]

\[
F_i(t) = F_i(t), 1 \leq t \leq T
\]

where \( W(t) \) records the number of the word model which gave the best score at frame \( t \), level \( \ell \), and \( F_i(t) \) records the backpointer of the best word model.

Each new level begins with the initial best probability at the preceding frame on the preceding level and increments the Viterbi score by matching the word models beginning at the new initial frame. This process is repeated through a number of levels equivalent to the maximum expected number of digits in any string (e.g., typically 7).

At the end of each level, a best string of size \( \ell \) words (1 \( \leq t \leq L \)) with probability \( P_i(T) \) is obtained by backtracking using the backpointer array \( F_i(t) \) to give the words in the string. The overall best string is the maximum of \( P_i(T) \) over all possible levels \( \ell \).

**C. Training the Word Models [59], [61]**

The key to success in connected word recognition is to derive word models from representative connected word strings. We have found that although the formal reestimation procedures developed in this paper work well, they are costly in terms of computation, and equivalently good parameter estimates can be obtained using a segmental-K-means procedure of the type discussed in Section VI. The only difference in the procedure, from the one discussed earlier, is that the training connected word strings are first segmented into individual digits, via a Viterbi alignment procedure, then each set of digits is segmented into states, and the vectors within each state are clustered into the best
M cluster solution. The segmental K-means reestimation of the HMM parameters is about an order of magnitude faster than the Baum–Welch reestimation procedure, and all our experimentation indicates that the resulting parameter estimates are essentially identical in that the resulting HMMs have essentially the same likelihood values. As such, the segmental K-means procedure was used to give all the results presented later in this section.

D. Duration Modeling for Connected Digits

There are two forms of durational information used in scoring connected digit sequences, namely word duration and state duration. The way in which word duration information is incorporated into the model scoring is as follows. At the end of each level, for each frame t, the accumulated probability \( P(t) \) is modified by determining the word duration \( r_w(t) \) as

\[
\tau_w(t) = t - F(t) + 1 \quad (123)
\]

and then multiplying the accumulated probability by the word duration probability, i.e.,

\[
P_i(t) = P(t) \cdot [\text{SR}(r_i(t), D_i, \sigma_i)]^{\gamma_{wo}} \cdot K_2 \quad (124)
\]

where \( \gamma_{wo} \) (set to 3.0) is a weighting factor on word durations, and \( K_2 \) is a normalization constant.

State duration probabilities are incorporated in a postprocessor. The level building recognizer provides multiple candidates at each level (by tracking multiple best scores at each frame of each level). Hence overall probability scores are obtained for \( R \) strings of length \( L \) digits, where \( R \) is the number of candidates per level (typically \( R = 2 \)). Each of the \( R \) strings is backtracked to give both individual words and individual states within the words. For an \( L \)-word string, if we denote the duration of state \( j \) at level \( l \) as \( \Delta_j(l) \), then, for each possible string, the postprocessor multiplies the overall accumulated probability \( P(T) \) by the state duration probabilities, giving

\[
P_i(T) = P(T) \cdot \prod_{t=1}^{L} \prod_{l=1}^{N} [p_{wij}^{(0)}(\Delta(l))]^{\gamma_{wo}} \cdot K_3 \quad (125)
\]

where \( \gamma_{wo} \) (set to 0.75) is a weighting factor on state durations, \( w(t) \) is the word at level \( t \), and \( K_3 \) is a normalization constant. The computation of (125) is performed on all \( R \) strings, and a reordered list of best strings is obtained. The incremental cost of the postprocessor computation is negligible compared to the computation to give \( P(T) \), and its performance has been shown to be comparable to the performance of the internal duration models.

E. Performance of the Connected Digit HMM Recognizer

The HMM-based connected digit recognizer has been trained and tested in 3 modes:

1) Speaker trained using 50 talkers (25 male, 25 female) each of whom provided a training set of about 300 connected digit strings and an independent testing set of 500 digit strings.

2) Multispeaker in which the training sets from the 50 talkers above were merged into a single large training set, and the testing sets were similarly merged. In this case a set of 6 HMMs per digit was used, where each HMM was derived from a subset of the training utterances.

3) Speaker independent based on the T1 training and testing databases. Both the training and testing sets had about 113 talkers (different ones were used in each set) and the talkers were divided into 22 dialectal groups. In this case a set of 4 HMMs per digit was used.

In each of the above databases there were variable length digit strings with from 1 to 7 digits per string.

The performance of the HMM connected digit recognizer, in these modes, is given in Table 2, where the entries in the table are average string error rates for cases in which the string length was unknown apriori (UL), and for cases in which the string length was known apriori (KL).

<table>
<thead>
<tr>
<th>Mode</th>
<th>Training Set</th>
<th>Testing Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speaker trained (50 talkers)</td>
<td>0.39</td>
<td>0.16</td>
</tr>
<tr>
<td></td>
<td>0.78</td>
<td>0.35</td>
</tr>
<tr>
<td>Multispeaker (50 talkers)</td>
<td>1.74</td>
<td>0.98</td>
</tr>
<tr>
<td></td>
<td>2.85</td>
<td>1.65</td>
</tr>
<tr>
<td>Speaker independent (112/113 talkers)</td>
<td>1.24</td>
<td>0.36</td>
</tr>
<tr>
<td></td>
<td>2.94</td>
<td>1.75</td>
</tr>
</tbody>
</table>

VIII. HMMs for Large Vocabulary Speech Recognition

Although HMMs have been successfully applied to problems in isolated and connected word recognition, the anticipated payoff of the theory, to problems in speech recognition, is in its application to large vocabulary speech recognition in which the recognition of speech is performed from basic speech units smaller than words. The research in this area far outweighs the research in any other area of speech processing and is far too extensive to discuss here. Instead, in this section we briefly outline the ideas of how HMMs have been applied to this problem.

In the most advanced systems (e.g., comparable to those under investigation at IBM, BBN, CMU and other places), the theory of HMMs has been applied to the representation of phoneme-like sub-words as HMMs; representation of words as HMMs; and representation of syntax as an HMM. To solve the speech recognition problem, a triply embedded network of HMMs must be used. This leads to an expanded network with an astronomical number of equivalent states; hence an alternative to the complete, exhaustive search procedure is required. Among the alternatives are the stack algorithm [7] and various forms of Viterbi beam searches [31]. These procedures have been shown to be capable of handling such large networks (e.g., 5000 words with an average word branching factor of 100) in an efficient and reliable manner. Details of these approaches are beyond the scope of this paper.

In another attempt to apply HMMs to continuous speech recognition, an ergodic HMM was used in which each state represented an acoustic-phonetic unit [47]. Hence about 40–50 states are required to represent all sounds of English. The model incorporated the variable duration feature in each state to account for the fact that vowel-like sounds
have vastly different durational characteristics than consonant-like sounds. In this approach, lexical access was used in conjunction with a standard pronouncing dictionary to determine the best matching word sequence from the output of the sub-word HMM. Again the details of this recognition system are beyond the scope of this paper. The purpose of this brief discussion is to point out the vast potential of HMMs for characterizing the basic processes of speech production; hence their applicability to problems in large vocabulary speech recognition.

A. Limitations of HMMs

Although use of HMM technology has contributed greatly to recent advances in speech recognition, there are some inherent limitations of this type of statistical model for speech. A major limitation is the assumption that successive observations (frames of speech) are independent, and therefore the probability of a sequence of observations \( \{O_1, O_2, \ldots, O_T\} \) can be written as a product of probabilities of individual observations, i.e.,

\[
P(O_1, O_2, \ldots, O_T) = \prod_{i=1}^{T} P(O_i).
\]

Another limitation is the assumption that the distributions of individual observation parameters can be well represented as a mixture of Gaussian or autoregressive densities. Finally the Markov assumption itself, i.e., that the probability of being in a given state at time \( t \) only depends on the state at time \( t - 1 \), is clearly inappropriate for speech sounds where dependencies often extend through several states. However, in spite of these limitations this type of statistical model has worked extremely well for certain types of speech recognition problems.

IX. Summary

In this paper we have attempted to present the theory of hidden Markov models from the simplest concepts (discrete Markov chains) to the most sophisticated models (variable duration, continuous density models). It has been our purpose to focus on physical explanations of the basic mathematics; hence we have avoided long, drawn out proofs and/or derivations of the key results, and concentrated primarily on trying to interpret the meaning of the math, and how it could be implemented in practice in real world systems. We have also attempted to illustrate some applications of the theory of HMMs to simple problems in speech recognition, and pointed out how the techniques could be (and have been) applied to more advanced speech recognition problems.

Acknowledgment

The author gratefully acknowledges the major contributions of several colleagues to the theory of HMMs in general, and to the presentation of this paper, in particular. A great debt is owed to Dr. J. Ferguson, Dr. A. Poritz, Dr. L. Liporace, Dr. A. Richter, and to Dr. F. Jelinek and the various members of the IBM group for introducing the speech world to the ideas behind HMMs. In addition Dr. S. Levinson, Dr. M. Sondhi, Dr. F. Juang, Dr. A. Demb, and Dr. Y. Ephraim have contributed significantly to both the theory of HMMs as well as the author’s perspective and knowledge as to how the theory is best applied to problems of speech recognition.

References

[23] A. P. Dempster, N. M. Laird, and D. B. Rubin, “Maximum like-


47. S. E. Levinson, “Continuous speech recognition by means of acoustic-phonetic classification obtained from a hidden Markov model,” in Proc. ICASSP ’87 (Dallas, TX), Apr. 1987.


Lawrence R. Rabiner (Fellow, IEEE) was born in Brooklyn, NY, on September 28, 1943. He received the S.B. and S.M. degrees, both in 1964, and the Ph.D. degree in electrical engineering, in 1967, all from the Massachusetts Institute of Technology, Cambridge, MA.


Dr. Rabiner is a member of Eta Kappa Nu, Sigma Xi, Tau Beta Pi, The National Academy of Engineering, and a Fellow of the Acoustical Society of America.
1 Notation for discrete HMM

$S = \{S_1, ..., S_N\}$. Set of possible hidden states.

$N$. Number of distinct hidden states.

$V = \{v_1, ..., v_M\}$. Set of possible external observations.

$M$. Number of distinct external observations.

$o = (o^1, ..., o^K)$. A sample of sequences of external observations (the training sample). Each element in $o$ is an entire sequence of observations (e.g., a word).

$K$. Number of sequences (e.g., words) in the training sample.

$o = (o_1, ..., o_T)$. A sequence of external observations (e.g., a word). If there is more than a sequence I use a superscript to denote the sequence.

$T$. The number of time steps in the sequence.

$q = (q^1, ..., q^K)$. Collection of sequences of internal states (internal state sequences that may correspond to each sequence in the training sequence).

$q = (q_1, ..., q_T)$. A sequence of internal states. If there is more than a sequence of interest I use a superscript to denote the sequence of interest.

$q_t$. A variable representing the internal state at time $t$. If there is more than a sequence I use a superscript (e.g., $q^4_2 = 3$ means that in sequence number 4 at time 2 we observe $S_3$).

$A; B$. A hidden Markov model as defined by its $A, B$ and $\pi$ matrices.

$a_{ij} = P(q_{t+1} = j | q_t = i)$. State transition probability for model $\lambda$.

$A = \{a_{ij}\}$. The $N \times N$ matrix of transition probabilities.

$b_j(k) = P(o_t = k | q_t = j)$. Emission probability for observation $v_k$ by state $S_j$.

$B = \{b_j(k)\}$. The $M \times N$ matrix of state to observations probabilities.

$\pi_i = P(q_1 = i)$. Initial state probability for model $\lambda$.

$\pi = \{\pi_i\}$. The vector of initial state probabilities.

$Q(\lambda, \tilde{\lambda}) = \sum_o P_o(q(0) \log P_{\lambda}(q(o))$. The auxiliary function maximized by the E-M algorithm. $\lambda$ represents the current model, $\tilde{\lambda}$ represents the new model under consideration.

$Q^t(\lambda, \tilde{\lambda})$ The E-M function restricted to the sequence $o^t$ from the training sample.

$\alpha_t(i) = P_x(o_1...o_t q_t = i)$. The forward variable for the sequence $o$ at time $t$ for state $i$. If there is more than one sequence of interest I use a superscript to denote the sequence.

$\beta_t(i) = P_x(o_{t+1}...o_T | q_t = i)$. The scaled backward variable for the sequence $o$ at time $t$ for state $i$. If there is more than one sequence of interest I use a superscript to denote the sequence.

$\hat{\alpha}_t(i)$. The scaled forward variable for the sequence $o$ at time $t$ for state $i$. If there is more than one sequence of interest I use a superscript to denote the sequence.

$\hat{\beta}_t(i)$. The scaled backward variable for the sequence $o$ at time $t$ for state $i$. If there is more than
than one sequence of interest I use a superscript to denote the sequence.

\[ c_1 \ldots c_T \]. The scaling coefficients in the scaled forward and backward algorithm for the sequence \( o' \). If there is more than one sequence of interest I use a superscript to denote the sequence.

\[ \gamma_t(i) = P_\lambda(q_t = i \mid o) \] If there is more than one sequence of interest I use a superscript to denote the sequence.

\[ \xi_t(i, j) = P_\lambda(q_{t+1} = j \mid o) \] If there is more than one sequence of interest I use a superscript to denote the sequence.

2 EM training with Discrete Observation Models

In this section we review two methods for training standard HMM models with discrete observations: E-M training and Viterbi training.

2.1 The E-M auxiliary function

Let \( \lambda \) represent the current model and \( \tilde{\lambda} \) represent a candidate model. Our objective is to make \( P_\lambda(o) \geq P_{\tilde{\lambda}}(o) \), or equivalently \( \log P_\lambda(o) \geq \log P_{\tilde{\lambda}}(o) \).

Due to the presence of stochastic constraints (e.g., \( a_{ij} \geq 0 \) and \( \sum_j a_{ij} = 1 \)) it turns out to be easier to maximize an auxiliary function \( Q(\cdot, \cdot) \) rather than to directly maximize \( \log P_\lambda(o) \).

The E-M auxiliary function is defined as follows:

\[ Q(\lambda, \tilde{\lambda}) = \sum_q P_\lambda(q \mid o) \log P_{\tilde{\lambda}}(q \mid o) \] (1)

Here we show that \( Q(\lambda, \tilde{\lambda}) \geq Q(\lambda, \lambda) \rightarrow \log P_\lambda(o) \geq \log P_{\tilde{\lambda}}(o) \).

For any model \( \lambda \) or \( \tilde{\lambda} \) it must be true that

\[ P_\lambda(o) = \frac{P_\lambda(oq)}{P_{\tilde{\lambda}}(q \mid o)} \] (2)

or \( \log P_\lambda(o) = \log P_\lambda(oq) - \log P_{\tilde{\lambda}}(q \mid o) \).

Also,

\[ \log P_{\tilde{\lambda}}(o) = \sum_q P_\lambda(q \mid o) \log P_{\tilde{\lambda}}(o) \] (3)

since \( \log P_{\tilde{\lambda}}(o) \) is a constant. Thus, from equation 2 it follows that

\[ \log P_\lambda(o) = \sum_q P_\lambda(q \mid o) \log P_{\tilde{\lambda}}(oq) - \sum_q P_\lambda(q \mid o) \log P_{\tilde{\lambda}}(q \mid o) \] (4)

\[ = Q(\lambda, \tilde{\lambda}) - \sum_q P_\lambda(q \mid o) \log P_{\tilde{\lambda}}(q \mid o) \] (5)

Applying the \( Q(\cdot, \cdot) \) function to \( (\lambda, \tilde{\lambda}) \) and to \( (\lambda, \lambda) \), it follows that,

\[ Q(\lambda, \tilde{\lambda}) - Q(\lambda, \lambda) = \log P_{\tilde{\lambda}}(o) - \log P_\lambda(o) - KL(\lambda, \tilde{\lambda}) \] (6)

where \( KL(\cdot, \cdot) \) is the Kullback-Leibler criterion (relative entropy) of the probability distribution \( P_\lambda(q \mid o) \) with respect to the probability distribution \( P_{\tilde{\lambda}}(q \mid o) \).
\[ KL(\lambda, \tilde{\lambda}) = \sum_q P_\lambda(q|o) log \frac{P_\lambda(q|o)}{P_\lambda(q|\tilde{o})} \]  

Rearranging terms,

\[ log P_\lambda(o) - log P_\lambda(\tilde{o}) = Q(\lambda, \tilde{\lambda}) - Q(\lambda, \lambda) + KL(\lambda, \tilde{\lambda}) \]  

and since the KL criterion is always positive, it follows that if

\[ Q(\lambda, \tilde{\lambda}) - Q(\lambda, \lambda) \geq 0 \]  

then

\[ log P_\lambda(o) log P_\lambda(\tilde{o}) \geq 0 \]  

2.2 The overall training sample E-M function

We defined the overall E-M function

\[ Q(\lambda, \tilde{\lambda}) = \sum_q P_\lambda(q|o) log P_\lambda(q|o) \]  

with \( o \) including the entire set of sequences in the training sample. Assuming that the sequences are independent, it follows that

\[ Q(\lambda, \tilde{\lambda}) = \sum_q P_\lambda(q|o) \sum_l log P_\lambda(q^l|o^l) = \sum_l \sum_q (log P_\lambda(q^l|o^l)) P_\lambda(q|o) \]  

And since each state sequence \( q^l \) depends only on the corresponding observation sequence \( o^l \) it follows that

\[ Q(\lambda, \tilde{\lambda}) = \sum_l \sum_q \sum_{q^l} (log P_\lambda(q^l|o^l)) \prod_{m=1}^{K} P_\lambda(q^m|o) \]  

where \( q - q^l = (q^1, \ldots, q^{l-1}, q^{l+1}, \ldots, q^K) \) represents an entire collection of sequences except for the sequence \( q^l \). Thus

\[ Q(\lambda, \tilde{\lambda}) = \sum_l \sum_q \sum_{q^l} (log P_\lambda(q^l|o^l)) P_\lambda(q^l|o^l) \prod_{m=1}^{K} P_\lambda(q^m|o) = \]  

\[ = \sum_l \sum_{q^l} (log P_\lambda(q^l|o^l)) P_\lambda(q^l|o^l) \sum_{q^m, m \neq l} \prod_{m=1}^{K} P_\lambda(q^m|o) \]  

and since

\[ \sum_{q^l, m \neq l} \prod_{m=1}^{K} P_\lambda(q^m|o) = 1 \]  

it follows that the E-M function can be decomposed into additive E-M functions, one per observation sequence in the training sample:

\[ Q(\lambda, \tilde{\lambda}) = \sum_l \sum_{q^l} P_\lambda(q^l|o^l) log P_\lambda(q^l|o^l) = \sum_l Q_l(\lambda, \tilde{\lambda}) \]
2.3 Maximizing the E-M function

For simplicity let us start with the case in which there is a single sequence. The results easily generalize to multiple sequences. Since we work with a single sequence we may drop the \( l \) superscript.

\[
Q(\lambda, \bar{\lambda}) = \sum_q P_\lambda(q|o) \log P_\lambda(qo) \tag{18}
\]

And since,

\[
P_\lambda(qo) = \bar{\pi}_{q_1} \bar{b}_{q_1}(o_1) \bar{a}_{q_1,q_2} \bar{b}_{q_2}(o_2) \bar{a}_{q_2,q_3} \cdots \tag{19}
\]

it follows that,

\[
Q(\lambda, \bar{\lambda}) = \sum_q P_\lambda(q|o) \log \bar{\pi}_{q_1} + \sum_{t=1}^{T_1} \sum_q P_\lambda(q|o) \log \bar{b}_{q_t}(o_t) + \sum_{t=1}^{T_1-1} \sum_q P_\lambda(q|o) \log \bar{a}_{q_t,q_{t+1}} \tag{20}
\]

The first term can be expressed as follows

\[
\sum_q P_\lambda(q|o) \log \bar{\pi}_{q_1} = \sum_j \log \bar{\pi}_j \sum_q P_\lambda(q|o) \delta(j, q_1) \tag{23}
\]

where \( \delta(j, q_1) \) tells us to include only those cases in which \( q_1 = j \). Therefore,

\[
\sum_q P_\lambda(q|o) \delta(j, q_1) = P_\lambda(q_1 = j|o) = \gamma_1(j) \tag{24}
\]

The second term can be expressed as follows

\[
\sum_{t=1}^{T_1} \sum_q P_\lambda(q|o) \log \bar{b}_{q_t}(o_t) = \sum_{t=1}^{T_1} \sum_i \sum_j \log \bar{b}_t(j) \sum_q P_\lambda(q|o) \delta(i, q_t) \delta(j, o_t) \tag{25}
\]

where \( \delta(i, q_t) \delta(j, o_t) \) tells us to include only those cases for which \( q_t = i \) and \( o_t = j \). Therefore,

\[
\sum_q P_\lambda(q|o) \delta(i, q_t) \delta(j, o_t) = P_\lambda(q_t = i|o) \delta(o_t, j) = \gamma_t(i) \delta(o_t, j) = \tag{26}
\]

The third term can be expressed as follows

\[
\sum_{t=1}^{T_1-1} \sum_q P_\lambda(q|o) \log \bar{a}_{q_t,q_{t+1}} = \sum_{t=1}^{T_1-1} \sum_i \sum_j \log \bar{a}_{ij} \sum_q P_\lambda(q|o) \delta(i, q_t) \delta(j, q_{t+1}) \tag{27}
\]

where \( \delta(i, q_t) \delta(j, q_{t+1}) \) tells us to include only those cases for which \( q_t = i \) and \( q_{t+1} = j \). Therefore,
\[
\sum_q P_q(q, q_t) \delta(i, q_t) \delta(j, q_{t+1}) = P_q(q_t = i \mid q_{t+1} = j, o) = \xi_t(i, j) \tag{28}
\]

Putting it together

\[
Q(\lambda, \bar{\lambda}) = \sum_j \gamma_1(j) \log \bar{\pi}_j +
\]

\[
+ \sum_i \sum_j \log \bar{b}_i(j) \left( \sum_{t=1}^{T_i} \gamma_t(i) \delta(o_t, j) \right) +
\]

\[
+ \sum_i \sum_j \log \bar{a}_{ij} \left( \sum_{t=1}^{T_i-1} \xi_t(i, j) \right) \tag{31}
\]

When there is more than a sequence, we just need to add up over sequences to obtain the overall \( Q(\cdot, \cdot) \) function

\[
Q(\lambda, \bar{\lambda}) = \sum_j \log \bar{\pi}_j \left( \sum_l \gamma_l^j(j) \right) +
\]

\[
+ \sum_i \sum_j \log \bar{b}_i(j) \left( \sum_{l=1}^{T_i} \sum_{t=1}^{T_l} \gamma_l^i(i) \delta(o_t^l, j) \right) +
\]

\[
+ \sum_i \sum_j \log \bar{a}_{ij} \left( \sum_{l=1}^{T_i-1} \sum_{t=1}^{T_l-1} \xi_t^l(i, j) \right) \tag{34}
\]

Note that the part of the overall \( Q(\lambda, \bar{\lambda}) \) function dependent on \( \bar{\pi}_j \) is of the form \( w_j \log x_j \) with \( x_j = \bar{\pi}_j \) and

\[
w_j = \sum_{l=1}^{K} \gamma_l^j(j) \tag{35}
\]

with constraints \( \sum_j x_j = 1 \), and \( x_j \geq 0 \).

Equivalently, the part of the overall \( Q(\lambda, \bar{\lambda}) \) function dependent of \( \bar{b}_i(j) \) is of the form \( w_j \log x_j \) with \( x_j = \bar{b}_i(j) \) and

\[
w_j = \sum_{l=1}^{K} \sum_{t=1}^{T_l} \gamma_l^i(j) \delta(o_t^l, j) \tag{36}
\]

with constraints \( \sum_j x_j = 1 \), and \( x_j \geq 0 \).

Finally, the part of the overall \( Q(\lambda, \bar{\lambda}) \) function dependent of \( \bar{a}_{ij} \) is also of the form \( w_j \log x_j \) with \( x_j = \bar{a}_{ij} \) and

\[
w_j = \sum_{l=1}^{K} \sum_{t=1}^{T_l-1} \xi_t^l(i, j) \tag{37}
\]

with constraints \( \sum_j x_j = 1 \), and \( x_j \geq 0 \).
It is easy to show that the maximum of a function of the form \( w_j \log x_j \) with constraints that \( x_j \geq 0 \) and \( \sum_j x_j = 1 \) is achieved for
\[
x_j = \frac{w_j}{\sum_j w_j}
\]
(38)

The parameters of the new model \( \tilde{\lambda} \) that maximize the overall \( Q(\lambda, \tilde{\lambda}) \) function easily follow:

\[
\tilde{\pi}_i = \frac{\sum_{t=1}^{K} \gamma_t(i)}{K} \\
\tilde{b}_t(j) = \frac{\sum_{t=1}^{K} \sum_{l=1}^{T_l} \gamma_t(i) \delta_t(o_l', j)}{\sum_{t=1}^{K} \sum_{l=1}^{T_l} \gamma_t(i)} \\
\tilde{a}_{ij} = \frac{\sum_{t=1}^{K} \sum_{l=1}^{T_l-1} \xi_t(i, j)}{\sum_{t=1}^{K} \sum_{l=1}^{T_l-1} \gamma_t(i)}
\]
(39)

2.4 Obtaining the E-M parameters from the scaled forward and backward algorithms

Section under construction

2.5 MM Training

MM training (Maximization Maximization) may be seen as an approximation to EM training. In MM training we basically substitute the Expected value operation by a Max operation thus the MM name. Other names for these algorithms are: Viterbi training (because we use the Viterbi algorithm to do the Max operation) and segmented K-means (K-means is a classical clustering method that belongs to the MM family).

In Viterbi-based decoding, the degree of match between a model \( \lambda \) and an observation sequence \( o' \) is defined as \( \rho(\lambda, o') = \max_{q} \log P_{\lambda}(q, o') = \log P_{\lambda}(q_1, o') \). We have seen how for a fixed model \( \lambda \), the Viterbi recurrence can be used to find \( \hat{q} \) and \( \rho(\lambda, o') \). When there is more than one training sequences they are assumed independent and \( \rho(\lambda, o) = \sum_{o'} \rho(\lambda, o') \). Thus, the optimal states can be found by applying Viterbi decoding independently for each of the training sequences.

In MM training the objective is to find an optimal point of \( \log P_{\lambda}(q, o') \) with the model \( \lambda \) and the state sequence \( q \) as optimizing variables.

To begin with, assume that Viterbi decoding has found the best sequence of hidden states for the current \( \lambda \) model: \( \hat{q} = (\hat{q}^1, ..., \hat{q}^K) \). Once we have \( \hat{q} \) we find a new model \( \tilde{\lambda} \) such that \( \log P_{\tilde{\lambda}}(\hat{q}, o) \geq \log P_{\lambda}(\hat{q}, o) \). To do so simply define a dummy model \( \tilde{\lambda} \) such that \( P_{\tilde{\lambda}}(q_1) = \delta(\hat{q}, o) \), thus the dummy model is such that only state sequence \( \hat{q} \) can co-occur with observation sequence \( o \).

For such model, \( P_{\tilde{\lambda}}(q_1 = j | o') = \delta(i, \hat{q}^i) \), \( P_{\tilde{\lambda}}(q_t = j | o') = \delta(i, \hat{q}_t) \), and \( P_{\tilde{\lambda}}(q_t = i | q_{t+1} = j | o') = \delta(i, \hat{q}_t) \delta(j, \hat{q}_{t+1}) \). Also note that \( Q(\lambda, \tilde{\lambda}) = \log P_{\lambda}(\hat{q}, o) \), the function we want
to optimize. Thus, substituting the $\hat{\lambda}$ parameters in the standard E-M formulas guarantees that $Q(\hat{\lambda}, \hat{\lambda}) \geq Q(\lambda, \lambda)$ or $\log P_\lambda(o, \tilde{q}) \geq \log P_\lambda(o, \hat{q})$.

Thus, the MM training rules are as follows:

$$\bar{\pi}_i = \frac{\sum_{l=1}^{K} \delta(i, \tilde{q}_l)}{K}$$

$$\bar{b}_i(j) = \frac{\sum_{k=1}^{K} \sum_{t=1}^{T} \delta(i, \tilde{q}_l) \delta(j, \tilde{q}_l) \delta(o, \tilde{q}_l)}{\sum_{r=1}^{K} \sum_{t=1}^{T} \delta(r, \tilde{q}_l)}$$

$$\bar{a}_{ij} = \frac{\sum_{k=1}^{K} \sum_{t=1}^{T-1} \delta(i, \tilde{q}_l) \delta(j, \tilde{q}_l \delta(o, \tilde{q}_l) + 1)}{\sum_{r=1}^{K} \sum_{t=1}^{T-1} \delta(i, \tilde{q}_l)} \tag{40}$$

Note that Viterbi decoding maximizes $\log P_\lambda(q, o)$ with respect to $q$ for a fixed model (the first M step) then we maximizes $\log P_\lambda(q, o)$ with respect to $\lambda$, for a fixed $q$ (the second M step). Since we are always maximizing with respect to some variables, $\log P_\lambda(q, o)$ can only increase and convergence to a local maximum is guaranteed.
3 Notation for Continuous Density Models

The notation for the continuous case is the same as the discrete case with the following additional terms.

$P$ Number of dimensions per observation (e.g. cepstral coefficients): $o_t = (o_{t1}, ..., o_{tP})$.

$M$. Number of clusters within a state.

$V = \{v_{11}, ..., v_{1M}, ..., v_{N1}, ..., v_{NM}\}$. Set of possible clusters of external observations (M clusters per state). Each cluster $v_{ij}$ is identified by a state index $i$ and a cluster index $j$.

$m_t$ variable identifying the cluster index of the cluster that occurred at time $t$. For example if cluster $v_{ik}$ occurred at time $t$ then $q_t = i$ and $m_t = k$.

$m = (m_1 ... m_t)$. A sequence of cluster indexes, one per time step.

$g_{ik} = P_{\lambda}(m_t = k | q_t = j)$. Emission probability (gain) of cluster $v_{ik}$ by state $S_j$.

$\phi(\cdot)$. A kernel function (e.g. Gaussian) to model the probability density of a cluster of observations produced by a state.

$b_{ik}(o_t) = P_{\lambda}(o_t | q_t = j)$. The centroid or prototype of cluster $v_{ik}$.

$\Sigma_{ik} = \begin{pmatrix} \sigma_{ik1}^2 & \sigma_{ik2} & ... & \sigma_{ikP} \\ \sigma_{ik21} & \sigma_{ik22} & ... & \sigma_{ik2P} \\ ... & ... & ... & ... \\ \sigma_{ikP1} & \sigma_{ikP2} & ... & \sigma_{ikP} \end{pmatrix}$ The covariance matrix of cluster $v_{ik}$.

$\sigma_{ikn}$. The variance of the $l^{th}$ dimension (e.g. cepstral) within cluster $v_{ik}$, a diagonal element of $\Sigma_{ik}$.

$\sigma_{iknm}$. The covariance between dimension $l$ and dimension $m$ within the cluster $v_{ik}$ an off-diagonal elements of $\Sigma_{ik}$. Usually assumed zero.

4 Continuous Observation Models

In this section we study the E-M and Viterbi training procedures for continuous observation HMMs.

4.1 Mixtures of Densities

In the discrete case the observations are discrete, represented by an integer. In the continuous case the observations at each time step are $P$-dimensional real-valued vectors (e.g. cepstrals coefficients). In our notation $o_t = (o_{t1}, ..., o_{tP})$.

The continuous observation model produces sequences of observations in the following way: At each time step the system generates a hidden state $q_t$ according to a state to state transition probability distribution $a_{q_{t-1}q_t}$. Once $q_t$ has been generated, the system generates a hidden cluster $m_t$ according to a state to cluster emission probability distribution $g_{n,m_t}$. Once the hidden cluster has been determined, an observation vector is produced probabilistically according to some kernel probability distribution (e.g. Multivariate Gaussian). We can think of the clusters as low level hidden states embedded within high level hidden states $q_t$. For example, the high level hidden states may represent phonemes and the low-level hidden clusters may represent acoustic categories within the same phoneme. For simplicity each state is assumed to have the same number of clusters ($M$) but the set of clusters is different from state to state. Thus, there is a total of $NxM$ clusters, $M$ per state.
We represent cluster k of state \( S_j \) as \( v_{jk} \) and we use the variable \( m_t \) to identify the cluster number within a state at time t. Thus if \( q_t = i \) and \( m_t = k \) it means that at time t cluster \( v_{jk} \) occurred.

If we know the cluster at time t, the probability density of a vector of continuous observations (e.g. cepstral coefficients) is modeled by a kernel function, usually a multivariate Gaussian

\[
P_X(o_t|v_{jk}) = P_X(o_t|q_t = j, m_t = k) = \phi(o_t, \mu_{jk}, \Sigma_{jk})
\]

(41)

Where \( \phi(\cdot) \) is the kernel function (e.g. multivariate Gaussian) \( \mu_{jk} = (\mu_{jk1}, \ldots, \mu_{jkP}) \) is a centroid or prototype that determines the position of the cluster in P-dimensional space, and

\[
\Sigma_{jk} = \begin{pmatrix}
\sigma_{jk1}^2 & \sigma_{jk12} & \cdots & \sigma_{jk1P} \\
\sigma_{jk21} & \sigma_{jk2}^2 & \cdots & \sigma_{jk2P} \\
\vdots & \vdots & \ddots & \vdots \\
\sigma_{jkP1} & \sigma_{jkP2} & \cdots & \sigma_{jkP}^2
\end{pmatrix}
\]

(42)

is a covariance matrix that determines the width and tilt of the cluster in P-dimensional space. The diagonal terms \( \{\sigma_{jk1}^2, \ldots, \sigma_{jkP}^2\} \) are the cluster variances for each dimension. They determine the spread of the cluster on each dimension. The off-diagonal elements are known as the cluster covariances and they determine the tilt of the cluster. For the Gaussian case, the kernel function is

\[
\phi(o_t, \mu_{jk}, \Sigma_{jk}) = \frac{1}{(2\pi)^{P/2}|\Sigma_{jk}|^{1/2}} e^{-d(o_t, \mu_{jk})}
\]

(43)

where \( |\Sigma_{jk}| \) is the determinant of the variance matrix, and \( d(o_t, \mu_{jk}) \) is known as the Mahalanobis distance between the observation and the kernel’s centroid.

\[
d(o_t, \mu_{jk}) = \frac{1}{2}(o_t - \mu_{jk})\Sigma_{jk}^{-1}(o_t - \mu_{jk})'
\]

(44)

Since the covariance matrices \( \Sigma_{jk} \) are symmetric, each kernel is defined by \( \frac{P(P+3)}{2} \) parameters (P for the centroids and \( P(P+1)/2 \) for the variances). In practice the off-diagonal variances are assumed zero, reducing the number of parameters per kernel to 2P. In such case the determinant \( |\Sigma_{jk}| \) is just the product of P scalar variances \( |\Sigma_{jk}| = \prod_{l=1}^{P} \sigma_{jk}^2 \), and the Mahalanobis distance becomes a scaled Euclidean distance:

\[
d(o_t, \mu_{jk}) = \frac{1}{2} \sum_{l=1}^{P} \frac{(o_t - \mu_{jk})^2}{\sigma_{jk}^2}
\]

(45)

Given a state \( S_j \), the system randomly chooses one of its M possible clusters with state to cluster emission probability \( P(m_t = k|q_t = j) \). This probability is assumed independent of t and thus it can be represented by a parameter with no time index. In our notation \( P(m_t = k|q_t = j) = g_{jk} \), the gain of the \( k^{th} \) cluster embedded in state \( S_j \).

Thus, the overall probability density of the observations generated by a state \( S_j \) is given by a weighted mixture of kernel functions.

\[
P_X(o_t|q_t = j) = \sum_{l=1}^{M} P(m_t = l|q_t = j)P(o_t|q_t = j, m_t = k) = \sum_{k=1}^{M} g_{jk} \phi(o_t, \mu_{jk}, \Sigma_{jk})
\]

(46)

(47)
4.2 Forward and backward variables

The un-scaled and the scaled algorithms work the same as in the discrete case. Only now the emission probability terms $b_j(o_t)$ are modeled by a mixture of densities.

$$b_j(o_t) = \sum_{k=1}^{M} g_{jk} \phi(o_t, \mu_{jk}, \Sigma_{jk})$$

(48)

4.3 EM Training

In the continuous case the clusters are low level hidden states $m_t$ embedded within high level hidden states $q_t$. Thus, the E-M function is defined over all possible $q_m$ sequences of high-level and low-level hidden states

$$Q(\lambda, \bar{\lambda}) = \sum_{q} \sum_{m} P(\lambda, q, m) \log P(\lambda, q, m)$$

(49)

and since

$$P(\lambda, q, m) = \bar{\pi}_{q_1} \bar{g}_{q_1, m_1} \phi(o_1, \bar{\mu}_{q_1, m_1}, \bar{\Sigma}_{q_1, m_1}), ..., \bar{a}_{qT-1} \bar{g}_{qT-1, m_{T-1}} \phi(o_T, \bar{\mu}_{qT, m_T}, \bar{\Sigma}_{qT, m_T})$$

(50)

it follows that

$$Q(\lambda, \bar{\lambda}) = \sum_{q} \sum_{m} P(\lambda, q, m) \log \bar{\pi}_{q_1} +$$

(51)

$$\sum_{t=1}^{T-1} \sum_{q} \sum_{m} P(\lambda, q, m) \log \bar{a}_{q_{t+1}} +$$

(52)

$$+ \sum_{t=1}^{T} \sum_{q} \sum_{m} P(\lambda, q, m) \log \bar{g}_{q_1, m}$$

(53)

$$+ \sum_{t=1}^{T} \sum_{q} \sum_{m} P(\lambda, q, m) \log \phi(o_t, \bar{\mu}_{q_t, m_t}, \bar{\Sigma}_{q_t, m_t})$$

(54)

Since the factors in the first two terms are independent of $m$ they simplify into

$$\sum_{q} \sum_{m} P(\lambda, q, m) \log \bar{\pi}_{q_1} = \sum_{q} P(\lambda, q) \log \bar{\pi}_{q_1}$$

(55)

and

$$\sum_{t=1}^{T-1} \sum_{q} \sum_{m} P(\lambda, q, m) \log \bar{a}_{q_{t+1}} = \sum_{t=1}^{T-1} \sum_{q} P(\lambda, q) \log \bar{a}_{q_{t+1}}$$

(56)

These terms are identical as in the discrete case and thus the same training rules for initial state probabilities and for state transition probabilities apply here.

To find the training formulas for the cluster gains, we focus on the part of $Q(\cdot)$ dependent on the gain terms. This part can be transformed as follows

$$\sum_{t=1}^{T} \sum_{q} \sum_{m} P(\lambda, q, m) \log \bar{g}_{q_t}(m_t) =$$

(57)
\[
T \sum_{t=1}^{N} \sum_{q=1}^{M} \sum_{m=1}^{q} \sum_{m} P_{X}(q_m|o) \log \bar{g}_{ik} \delta(i, q_t) \delta(j, m_t)
\]

(58)

\[
= \sum_{t=1}^{T} \sum_{i=1}^{N} \sum_{k=1}^{M} P_{X}(q_t = i, m_t = k|o) \log \bar{g}_{ik}
\]

(59)

Thus the part of \(Q(\cdot)\) that depends on \(\bar{g}_{ik}\) is of the form \(w_j \log x_j\) with \(x_j = \bar{g}_{ik}\) and

\[
w_j = \sum_{t=1}^{T} P_{X}(q_t = i, m_t = k|o)
\]

(60)

with constraints \(\sum_j x_j = 1\) and \(x_j \leq 0\), with maximum achieved for

\[
x_j = \frac{w_j}{\sum_j w_j}
\]

(61)

Thus

\[
\bar{g}_{ik} = \frac{\sum_{t=1}^{T} P_{X}(q_t = i, m_t = k|o)}{\sum_{i=1}^{T} \sum_{k=1}^{M} P_{X}(q_t = i, m_t = k|o)} = \frac{\sum_{t=1}^{T} P_{X}(q_t = i, m_t = k|o)}{\sum_{t=1}^{T} P_{X}(q_t = i|o)}
\]

(62)

To find the learning rules for the centroids and variances we focus on the part of \(Q(\cdot)\) that depends on the cluster centroids and variances, which is given by the following expression:

\[
\sum_{t=1}^{T} \sum_{q=1}^{M} \sum_{m} P_{X}(q_m|o) \log \phi(o_t, \mu_{q,m}, \Sigma_{q,m})
\]

(63)

This expression can be transformed as follows

\[
\sum_{t=1}^{T} \sum_{q=1}^{M} \sum_{m} P_{X}(q_m|o) \log \phi(o_t, \mu_{q,m}, \Sigma_{q,m}) =
\]

(64)

\[
= \sum_{t=1}^{T} \sum_{i=1}^{N} \sum_{k=1}^{M} \sum_{q=1}^{m} P_{X}(q_m|o) \log \phi(o_t, \mu_{q,m}, \Sigma_{q,m}) \delta(i, q_t) \delta(k, m_t)
\]

(65)

\[
= \sum_{t=1}^{T} \sum_{i=1}^{N} \sum_{k=1}^{M} P_{X}(q_t = i, m_t = k|o) \log \phi(o_t, \mu_{q,k}, \Sigma_{q,k})
\]

(66)

Thus, at a maximum,

\[
\frac{\partial}{\partial \mu_{ikn}} \sum_{t=1}^{T} \sum_{i=1}^{N} \sum_{k=1}^{M} P_{X}(q_t = i, m_t = k|o) \log \phi(o_t, \mu_{q,k}, \Sigma_{q,k}) = 0
\]

(67)

and since

\[
\frac{\partial \log \phi(o_t, \mu_{q,k}, \Sigma_{q,k})}{\partial \mu_{ikn}} = \frac{1}{\sigma_{ik}^2} (o_{tl} - \bar{\mu}_{ikn})
\]

(68)

it follows that at a maximum

\[
\sum_{t=1}^{T} P_{X}(q_t = i, m_t = k|o) (o_{tl} - \bar{\mu}_{ikn}) = 0
\]

(69)
or

$$\tilde{\mu}_{ikn} = \frac{\sum_{t=1}^{T} P_\lambda(q_t = i \ m_t = k | o) \ o_{tn}}{\sum_{t=1}^{T} P_\lambda(q_t = i | o)} \quad (70)$$

A similar argument can be made for the diagonal variance $\sigma_{ik}$ In this case

$$\frac{\partial \log \phi(o_t, \tilde{\mu}_{ik}, \tilde{\Sigma}_{ik})}{\partial \sigma_{ik}^2} = -\frac{1}{2\sigma_{ik}^2} (1 - \frac{(o_{tn} - \tilde{\mu}_{ikn})^2}{\sigma_{ikn}^2}) \quad (71)$$

Thus, at a maximum

$$\sum_{t=1}^{T} P_\lambda(q_t = i \ m_t = k | o)(1 - \frac{(o_{tn} - \tilde{\mu}_{ikn})^2}{\sigma_{ikn}^2}) = 0 \quad (72)$$

from which the re-estimation formula easily follows:

$$\tilde{\sigma}_{ikn}^2 = \frac{\sum_{t=1}^{T} P_\lambda(q_t = i \ m_t = k | o)(o_{tn} - \tilde{\mu}_{ikn})^2}{\sum_{t=1}^{T} P_\lambda(q_t = i | o)} \quad (73)$$

Training for the mixture gains, mixture centroids, and mixture variances requires the $P(q_t = j \ m_t = k | o)$ terms, for $t = 1...T, j = 1..N, k = 1...M$. To obtain these terms note the following:

$$P(q_t = j \ m_t = k | o) = P(q_t = j | o)P(m_t = k | q_t = j o) =$$

$$= P(q_t = j | o)P(m_t = k | q_t = j o) \quad (74)$$

$$= P(q_t = j | o)P(o_t m_t = k | q_t = j)$$

$$= P(q_t = j | o) \frac{P(m_t = k | q_t = j) P(o_t | q_t = j m_t = k)}{P(o_t | q_t = j)} \quad (75)$$

Thus

$$P(q_t = j m_t = k | o) = P(q_t = j | o) \frac{g_{jk} \phi(o_t, \mu_{ik}, \Sigma_{ik})}{b_j(o_t)} \quad (78)$$

As in the discrete case, the $P(q_t = j | o)$ can be obtained through the scaled feed-forward algorithm.

For the case with multiple training sequences the overall $Q(\cdot)$ decomposes into additive $Q^l(\cdot)$, one per training sequence. As a consequence we have to add in the numerator and denominator of the training formulas the effects of each training sequence.

Summarizing, the E-M learning rules for the mixture of Gaussian densities case with diagonal covariance matrices are as follows:
\[ \tilde{\pi}_i = \frac{\sum_{k=1}^K \lambda_\alpha(q_t = j|\alpha')}{\sum_{k=1}^K \lambda_\alpha(q_{t+1} = j|\alpha')} \]

\[ \tilde{\alpha}_{ij} = \frac{\sum_{k=1}^K \sum_{t=1}^{T_l-1} \lambda_\alpha(q_t = i, q_{t+1} = j|\alpha')}{\sum_{k=1}^K \sum_{t=1}^{T_l-1} \lambda_\alpha(q_t = i|\alpha')} \]

\[ \tilde{\gamma}_{ik} = \frac{\sum_{k=1}^K \sum_{t=1}^{T_l-1} \lambda_\alpha(q_{t+1} = i, m_t = k|\alpha')}{\sum_{t=1}^{T_l-1} \lambda_\alpha(q_t = i|\alpha')} \]

\[ \tilde{\mu}_{ikn} = \frac{\sum_{k=1}^K \sum_{t=1}^{T_l-1} \lambda_\alpha(q_{t+1} = i, m_t = k|\alpha') (\gamma_{tn} - \mu_{ikn})^2}{\sum_{t=1}^{T_l-1} \lambda_\alpha(q_t = i|\alpha')} \]

\[ \tilde{\sigma}_{ikn}^2 = \sum_{k=1}^K \sum_{t=1}^{T_l-1} \lambda_\alpha(q_{t+1} = i, m_t = k|\alpha') (\gamma_{tn} - \mu_{ikn})^2 \]

\[ P(q_t = i, m_t = k|\alpha) = \frac{P(q_t = i|\alpha) \frac{\phi(\alpha, \mu_{ik}, \Sigma_{ik})}{b_i(\alpha_t)}}{b_i(\alpha_t)} \]

\[ b_i(\alpha_t) = \sum_{k=1}^M g_{ik} \phi(\alpha_t, \mu_{ik}, \Sigma_{ik}) \]

\[ \phi(\alpha_t, \mu_{ik}, \Sigma_{ik}) = \frac{1}{(2\pi)^{T/2} |\Sigma_{ik}|^{1/2}} e^{-d(\alpha_t, \mu_{ik})} \]

\[ d(\alpha_t, \mu_{ik}) = \frac{1}{2} \sum_{n=1}^P (\gamma_{tn} - \mu_{ikn})^2 \]

\[ P_\lambda(q_t = i|\alpha') = \frac{\tilde{\alpha}_{ij}(t) \tilde{\beta}_{ij}(t)}{\sum \tilde{\alpha}_{ij}(t) \tilde{\beta}_{ij}(t)} \]

\[ P_\lambda(q_t = i, q_{t+1} = j|\alpha') = \tilde{\alpha}_{ij}(t) a_{ij} \tilde{\beta}_{ij}(t+1) b_j(\alpha'_{t+1}) \]

where \( l = 1, \ldots, K \) indexes the training sequence, \( t = 1, \ldots, T_l \) indexes the time within a training sequence, \( i = 1, \ldots, N \) and \( j = 1, \ldots, N \) index the hidden state, \( k = 1, \ldots, M \) indexes the cluster embedded within a state, and \( n = 1, \ldots, P \) indexes the dimension of the continuous vector of observations (e.g., the cepstral coefficient). The \( P(q_t = jm_t = k|\alpha) \), \( P_\lambda(q_t = i|\alpha') \) and \( P_\lambda(q_t = i, q_{t+1} = j|\alpha') \) terms are obtained from the scaled forward-backward algorithms according to the following formulas:
Thus, the Viterbi training rules are as follows:

4.4 Viterbi decoding

We can use Viterbi decoding to find the best possible sequence of high level hidden states \( q \) and low level clusters \( m \). There are two approaches to this problem. One approach attempts to find simultaneously the best joint sequence of high-level and low-level states. Thus the goal is to find \( \hat{q} \hat{m} = \arg \max_{q,m} P_{\lambda}(q,m|o) \). The second approach first finds the best possible sequence of high level states \( \hat{q} = \arg \max_{q} P_{\lambda}(q|o) \) and once \( \hat{q} \) has been found, \( \hat{m} \) is defined as \( \hat{m} = \arg \max_{m} P_{\lambda}(\hat{m}|\hat{q},o) \). The two approaches do not necessarily yield the same results. The second approach is the standard in the literature.

For the second, most used version of Viterbi decoding, the same Viterbi recurrence as in the discrete case applies but using the continuous version of \( b_t(o_t) \). Once we have \( \hat{q}_t \), the desired \( \hat{m}_t \) is simply the cluster within \( S_{\hat{q}_t} \) which is closest (in Mahalanobis distance) to \( o_t \).

4.5 Viterbi training

The objective in Viterbi training (also known as segmental k-means) is to find an optimal point (local maximum) of \( \log P_{\lambda}(o \hat{q} \hat{m}) \) with \( q \) and \( \lambda \) being the optimizing variables. It does not matter how \( \hat{q} \hat{m} \) are found as long as a consistent procedure is used throughout training. As in the discrete case we define a dummy model \( \lambda \) such that \( P_{\lambda}(q^t, m^t|o) = \delta(q^t, q^t) \delta(m^t, m^t) \). For such model, \( P_{\lambda}(\hat{q}_t = j | o^t) = \delta(i, \hat{q}_t) \). \( P_{\lambda}(\hat{q}_t = i | o^t) = \delta(i, \hat{q}_t) \delta(j, \hat{q}_{t+1}) \), and \( P_{\lambda}(\hat{q}_t = i \hat{m}_t = k | o^t) = \delta(i, \hat{q}_t) \delta(k, \hat{m}_t) \).

As in the discrete case note that \( Q(\lambda, \lambda) = \log P_{\lambda}(o \hat{q} \hat{m}) \), the function we want to optimize. Thus, substituting the \( \lambda \) parameters in the standard E-M formulas guarantees that \( Q(\lambda, \lambda) \geq Q(\hat{\lambda}, \lambda) \), and thus \( \log P_{\lambda}(o \hat{q} \hat{m}) \geq \log P_{\lambda}(o \hat{q} \hat{m}) \).

Thus, the Viterbi training rules are as follows:

\[
\begin{align*}
\bar{\pi}_i &= \frac{\sum_{k=1}^{K} \delta(i, \hat{q}_1)}{K} \\
\bar{a}_{ij} &= \frac{\sum_{k=1}^{K} \sum_{l=1}^{T-1} \delta(i, \hat{q}_l) \delta(j, \hat{q}_{l+1})}{\sum_{k=1}^{K} \sum_{l=1}^{T-1} \delta(i, \hat{q}_l)} \\
\bar{g}_{ik} &= \frac{\sum_{k=1}^{K} \sum_{l=1}^{T-1} \delta(i, \hat{q}_l) \delta(k, \hat{m}_l)}{\sum_{j=1}^{K} \sum_{l=1}^{T-1} \delta(i, \hat{q}_l)} \\
\bar{\mu}_{ikn} &= \frac{\sum_{k=1}^{K} \sum_{l=1}^{T-1} \delta(i, \hat{q}_l) \delta(k, \hat{m}_l) \alpha_{tn}}{\sum_{l=1}^{T-1} \delta(i, \hat{q}_l)} \\
\bar{\sigma}^2_{ikn} &= \frac{\sum_{k=1}^{K} \sum_{l=1}^{T-1} \delta(i, \hat{q}_l) \delta(k, \hat{m}_l) (\alpha_{tn} - \bar{\mu}_{ikn})^2}{\sum_{l=1}^{T-1} \delta(i, \hat{q}_l)}
\end{align*}
\]

Since Viterbi decoding maximizes \( P_{\lambda}(q, m, o) \) with respect to \( q \) and \( m \) and Viterbi training maximizes \( P_{\lambda}(q, m, o) \) with respect to \( \lambda \), repeatedly applying Viterbi decoding and
Viterbi training, can only make $P_x(q, m, o)$ increase and convergence to a local maximum is guaranteed.
5 Factored Sampling Methods for Continuous State Models

Many recognition problems can be framed in terms of inferring something about $q_t$ the internal state of a system, based on a sequence of observations $o = o_1 \cdots o_t$. These inferences are in many cases based on estimates of $p(q_t|o_1 \cdots o_t)$. When the states are discrete and countable, these conditional state probabilities can be obtained using the forwards algorithm. However, the algorithm cannot be used when the states are continuous. In such case, direct sampling methods are appropriate. Here is an example of how these methods work.

We start with a sensor model: $p(o_t|q_t)$ and a Markovian state dynamics model $p(q_{t+1}|q_t)$. Our goal is to obtain estimates of $p(q_t|o_t)$ for all $t$.

1. Recursion
   Assume we have an estimate $\hat{p}(q_t|o_t)$. Our goal is to update that estimate for the next time step $p(q_{t+1}|o_{t+1})$.
   
   (a) First we draw a random sample $X$ from $\hat{p}(q_t|o_t)$. This sample will implicitly define a re-estimation of $p(q_t|o_t)$ in terms of a mixture of delta functions:
   $$\hat{p}(q_t|o_t) = \frac{1}{N} \sum_{i=1}^{N} \delta(q_t, x_i)$$
   (b) For each observation $x_i$ we obtain another random observation $y_i$ using the state dynamics $p(q_{t+1}|q_t = x_i)$. The new sample $Y = \{y_1 \cdots y_n\}$ implicitly defines our estimates of $p(q_{t+1}|o_t)$.
   $$\hat{p}(q_{t+1}|o_t) = \frac{1}{N} \sum_{i=1}^{N} \delta(q_{t+1}, y_i)$$
   (c) We know that
   $$p(q_{t+1}|o_1 \cdots o_t) = \frac{p(o_1 \cdots o_t)}{p(o_1 \cdots o_{t+1})} p(q_{t+1}|o_1 \cdots o_{t+1}) p(o_1 | o_1)$$
   The fraction is a constant $K(o_{t+1})$ independent of $q_{t+1}$, we already have an estimate of $p(q_{t+1}|o_t)$ so we just need to weight it by $p(o_1 | o_1)$. We obtain another random observation
   $$\hat{p}(q_{t+1}|o_t) = K(o_{t+1}) \frac{1}{N} \sum_{i=1}^{N} \delta(q_{t+1}, y_i) p(o_{t+1}|q_{t+1}) =$$
   $$\frac{1}{N} \sum_{i=1}^{N} \delta(q_{t+1}, y_i) p(o_{t+1}|q_{t+1})$$
   We can now use $\hat{p}(q_{t+1}|o_{t+1})$ to estimate parameters like the mean or the variance of the distribution. More generally,
   $$\hat{\omega} = \int dq_{t+1} Q(p(q_{t+1}|o_{t+1}), q_{t+1})$$

2. Initialization
   The initialization step is basically the same as the recursion step only that instead of using the state transition probabilities we use the initial state probabilities
   (a) Obtain a sample of $N$ random states: $X = \{x_1 \cdots x_N\}$ from the initial state probability function $\pi(\cdot)$. These $N$ samples will implicitly work as our estimate of the initial state probability.
   $$\hat{p}(q_1) = \frac{1}{N} \sum_{i=1}^{N} \delta(q_1, y_i)$$
   (b) Weight each observation by the sensor probability $p(o_1|q_1 = y_i)$. This defines our initial distribution estimates
   $$\hat{p}(q_0|o_1) = \frac{1}{N} \sum_{i=1}^{N} \delta(q_0, y_i) p(o_0|q_0)$$
A Tutorial on Support Vector Machines for Pattern Recognition

CHRISTOPHER J.C. BURGES
Bell Laboratories, Lucent Technologies

burges@lucent.com

Editor: Usama Fayyad

Abstract. The tutorial starts with an overview of the concepts of VC dimension and structural risk minimization. We then describe linear Support Vector Machines (SVMs) for separable and non-separable data, working through a non-trivial example in detail. We describe a mechanical analogy, and discuss when SVM solutions are unique and when they are global. We describe how support vector training can be practically implemented, and discuss in detail the kernel mapping technique which is used to construct SVM solutions which are nonlinear in the data. We show how Support Vector machines can have very large (even infinite) VC dimension by computing the VC dimension for homogeneous polynomial and Gaussian radial basis function kernels. While very high VC dimension would normally bode ill for generalisation performance, and while at present there exists no theory which shows that good generalisation performance is guaranteed for SVMs, there are several arguments which support the observed high accuracy of SVMs, which we review. Results of some experiments which were inspired by these arguments are also presented. We give numerous examples and proofs of most of the key theorems. There is new material, and I hope that the reader will find that even old material is cast in a fresh light.

Keywords: Support Vector Machines, Statistical Learning Theory, VC Dimension, Pattern Recognition

1. Introduction

The purpose of this paper is to provide an introductory yet extensive tutorial on the basic ideas behind Support Vector Machines (SVMs). The books (Vapnik, 1995; Vapnik, 1998) contain excellent descriptions of SVMs, but they leave room for an account whose purpose from the start is to teach. Although the subject can be said to have started in the late seventies (Vapnik, 1979), it is only now receiving increasing attention, and so the time appears suitable for an introductory review. The tutorial dwells entirely on the pattern recognition problem. Many of the ideas there carry directly over to the cases of regression estimation and linear operator inversion, but space constraints precluded the exploration of these topics here.

The tutorial contains some new material. All of the proofs are my own versions, where I have placed a strong emphasis on their being both clear and self-contained, to make the material as accessible as possible. This was done at the expense of some elegance and generality: however generality is usually easily added once the basic ideas are clear. The longer proofs are collected in the Appendix.

By way of motivation, and to alert the reader to some of the literature, we summarize some recent applications and extensions of support vector machines. For the pattern recognition case, SVMs have been used for isolated handwritten digit recognition (Cortes and Vapnik, 1995; Schölkopf, Burges and Vapnik, 1995; Schölkopf, Burges and Vapnik, 1996; Burges and Schölkopf, 1997), object recognition (Blanz et al., 1996), speaker identification (Schmidt, 1996), charmed quark detection, face detection in images (Osuna, Freund and Girosi, 1997), and text categorization (Joachims, 1997). For the regression estimation case, SVMs have been compared on benchmark time series prediction tests (Müller et al., 1997; Mukherjee, Osuna and Girosi, 1997), the Boston housing problem (Drucker et al., 1997), and (on artificial data) on the PET operator inversion problem (Vapnik, Golowich...
and Smola, 1996). In most of these cases, SVM generalization performance (i.e. error rates on test sets) either matches or is significantly better than that of competing methods. The use of SVMs for density estimation (Weston et al., 1997) and ANOVA decomposition (Stitson et al., 1997) has also been studied. Regarding extensions, the basic SVMs contain no prior knowledge of the problem (for example, a large class of SVMs for the image recognition problem would give the same results if the pixels were first permuted randomly (with each image suffering the same permutation), an act of vandalism that would leave the best performing neural networks severely handicapped) and much work has been done on incorporating prior knowledge into SVMs (Schölkopf, Burges and Vapnik, 1996; Schölkopf et al., 1998a; Burges, 1998). Although SVMs have good generalization performance, they can be abysmally slow in test phase, a problem addressed in (Burges, 1996; Osuna and Girosi, 1998). Recent work has generalized the basic ideas (Smola, Schölkopf and Müller, 1998a; Smola and Schölkopf, 1998), shown connections to regularization theory (Smola, Schölkopf and Müller, 1998b; Girosi, 1998; Wahba, 1998), and shown how SVM ideas can be incorporated in a wide range of other algorithms (Schölkopf, Smola and Müller, 1998b; Schölkopf et al, 1998c). The reader may also find the thesis of (Schölkopf, 1997) helpful.

The problem which drove the initial development of SVMs occurs in several guises - the bias variance tradeoff (Geman and Bienenstock, 1992), capacity control (Guyon et al., 1992), overfitting (Montgomery and Peck, 1992) - but the basic idea is the same. Roughly speaking, for a given learning task, with a given finite amount of training data, the best generalization performance will be achieved if the right balance is struck between the accuracy attained on that particular training set, and the “capacity” of the machine, that is, the ability of the machine to learn any training set without error. A machine with too much capacity is like a botanist with a photographic memory who, when presented with a new tree, concludes that it is not a tree because it has a different number of leaves from anything she has seen before; a machine with too little capacity is like the botanist’s lazy brother, who declares that if it’s green, it’s a tree. Neither can generalize well. The exploration and formalization of these concepts has resulted in one of the shining peaks of the theory of statistical learning (Vapnik, 1979).

In the following, bold typeface will indicate vector or matrix quantities; normal typeface will be used for vector and matrix components and for scalars. We will label components of vectors and matrices with Greek indices, and label vectors and matrices themselves with Roman indices. Familiarity with the use of Lagrange multipliers to solve problems with equality or inequality constraints is assumed2.

2. A Bound on the Generalization Performance of a Pattern Recognition Learning Machine

There is a remarkable family of bounds governing the relation between the capacity of a learning machine and its performance3. The theory grew out of considerations of under what circumstances, and how quickly, the mean of some empirical quantity converges uniformly, as the number of data points increases, to the true mean (that which would be calculated from an infinite amount of data) (Vapnik, 1979). Let us start with one of these bounds.

The notation here will largely follow that of (Vapnik, 1995). Suppose we are given \( l \) observations. Each observation consists of a pair: a vector \( x_i \in \mathbb{R}^n \), \( i = 1, \ldots, l \) and the associated “truth” \( y_i \), given to us by a trusted source. In the tree recognition problem, \( x_i \) might be a vector of pixel values (e.g., \( n = 256 \) for a 16x16 image), and \( y_i \) would be 1 if the image contains a tree, and -1 otherwise (we use -1 here rather than 0 to simplify subsequent formulae). Now it is assumed that there exists some unknown probability distribution
\(P(x, y)\) from which these data are drawn, i.e., the data are assumed “iid” \(\text{ (independently drawn and identically distributed)}\). \(\text{(We will use } P \text{ for cumulative probability distributions, and } p \text{ for their densities). Note that this assumption is more general than associating a fixed } y \text{ with every } x: \text{ it allows there to be a distribution of } y \text{ for a given } x. \text{ In that case, the trusted source would assign labels } y_i \text{ according to a fixed distribution, conditional on } x_i. \text{ However, after this Section, we will be assuming fixed } y \text{ for given } x.\)

Now suppose we have a machine whose task it is to learn the mapping \(x_i \rightarrow y_i\). The machine is actually defined by a set of possible mappings \(x \rightarrow f(x, \alpha)\), where the functions \(f(x, \alpha)\) themselves are labeled by the adjustable parameters \(\alpha\). The machine is assumed to be deterministic: for a given input \(x\), and choice of \(\alpha\), it will always give the same output \(f(x, \alpha)\). A particular choice of \(\alpha\) generates what we will call a “trained machine.” Thus, for example, a neural network with fixed architecture, with \(\alpha\) corresponding to the weights and biases, is a learning machine in this sense.

The expectation of the test error for a trained machine is therefore:

\[
R(\alpha) = \int \frac{1}{2} |y - f(x, \alpha)| dP(x, y) \tag{1}
\]

Note that, when a density \(p(x, y)\) exists, \(dP(x, y)\) may be written \(p(x, y) dx \, dy\). This is a nice way of writing the true mean error, but unless we have an estimate of what \(P(x, y)\) is, it is not very useful.

The quantity \(R(\alpha)\) is called the expected risk, or just the risk. Here we will call it the actual risk, to emphasize that it is the quantity that we are ultimately interested in. The “empirical risk” \(R_{\text{emp}}(\alpha)\) is defined to be just the measured mean error rate on the training set \((\text{for a fixed, finite number of observations})\):

\[
R_{\text{emp}}(\alpha) = \frac{1}{2l} \sum_{i=1}^{l} |y_i - f(x_i, \alpha)|. \tag{2}
\]

Note that no probability distribution appears here. \(R_{\text{emp}}(\alpha)\) is a fixed number for a particular choice of \(\alpha\) and for a particular training set \(\{x_i, y_i\}\).

The quantity \(\frac{1}{2} |y_i - f(x_i, \alpha)|\) is called the loss. For the case described here, it can only take the values 0 and 1. Now choose some \(\eta\) such that \(0 \leq \eta \leq 1\). Then for losses taking these values, with probability \(1 - \eta\), the following bound holds (Vapnik, 1995):

\[
R(\alpha) \leq R_{\text{emp}}(\alpha) + \sqrt{\left( h \log \left( \frac{2l}{h} \right) + 1 \right) \frac{- \log(\eta/4)}{l}} \tag{3}
\]

where \(h\) is a non-negative integer called the Vapnik Chervonenkis (VC) dimension, and is a measure of the notion of capacity mentioned above. In the following we will call the right hand side of Eq. (3) the “risk bound.” We depart here from some previous nomenclature: the authors of (Guyon et al., 1992) call it the “guaranteed risk”, but this is something of a misnomer, since it is really a bound on a risk, not a risk, and it holds only with a certain probability, and so is not guaranteed. The second term on the right hand side is called the “VC confidence.”

We note three key points about this bound. First, remarkably, it is independent of \(P(x, y)\). It assumes only that both the training data and the test data are drawn independently according to \textit{some} \(P(x, y)\). Second, it is usually not possible to compute the left hand side. Third, if we know \(h\), we can easily compute the right hand side. Thus given several
different learning machines (recall that “learning machine” is just another name for a family of functions $f(x, \alpha)$), and choosing a fixed, sufficiently small $\eta$, by then taking that machine which minimizes the right hand side, we are choosing that machine which gives the lowest upper bound on the actual risk. This gives a principled method for choosing a learning machine for a given task, and is the essential idea of structural risk minimization (see Section 2.6). Given a fixed family of learning machines to choose from, to the extent that the bound is tight for at least one of the machines, one will not be able to do better than this. To the extent that the bound is not tight for any, the hope is that the right hand side still gives useful information as to which learning machine minimizes the actual risk. The bound not being tight for the whole chosen family of learning machines gives critics a justifiable target at which to fire their complaints. At present, for this case, we must rely on experiment to be the judge.

2.1. The VC Dimension

The VC dimension is a property of a set of functions $\{f(\alpha)\}$ (again, we use $\alpha$ as a generic set of parameters: a choice of $\alpha$ specifies a particular function), and can be defined for various classes of function $f$. Here we will only consider functions that correspond to the two-class pattern recognition case, so that $f(x, \alpha) \in \{-1, 1\} \forall x, \alpha$. Now if a given set of $l$ points can be labeled in all possible $2^l$ ways, and for each labeling, a member of the set $\{f(\alpha)\}$ can be found which correctly assigns those labels, we say that that set of points is shattered by that set of functions. The VC dimension for the set of functions $\{f(\alpha)\}$ is defined as the maximum number of training points that can be shattered by $\{f(\alpha)\}$. Note that, if the VC dimension is $h$, then there exists at least one set of $h$ points that can be shattered, but it in general it will not be true that every set of $h$ points can be shattered.

2.2. Shattering Points with Oriented Hyperplanes in $\mathbb{R}^n$

Suppose that the space in which the data live is $\mathbb{R}^2$, and the set $\{f(\alpha)\}$ consists of oriented straight lines, so that for a given line, all points on one side are assigned the class 1, and all points on the other side, the class $-1$. The orientation is shown in Figure 1 by an arrow, specifying on which side of the line points are to be assigned the label 1. While it is possible to find three points that can be shattered by this set of functions, it is not possible to find four. Thus the VC dimension of the set of oriented lines in $\mathbb{R}^2$ is three.

\[\text{Figure 1. Three points in } \mathbb{R}^2, \text{ shattered by oriented lines.}\]
Let's now consider hyperplanes in $\mathbb{R}^n$. The following theorem will prove useful (the proof is in the Appendix):

**Theorem 1** Consider some set of $m$ points in $\mathbb{R}^n$. Choose any one of the points as origin. Then the $m$ points can be shattered by oriented hyperplanes if and only if the position vectors of the remaining points are linearly independent.

**Corollary:** The VC dimension of the set of oriented hyperplanes in $\mathbb{R}^n$ is $n + 1$, since we can always choose $n + 1$ points, and then choose one of the points as origin, such that the position vectors of the remaining $n$ points are linearly independent, but can never choose $n + 2$ such points (since no $n + 1$ vectors in $\mathbb{R}^n$ can be linearly independent).

An alternative proof of the corollary can be found in (Anthony and Biggs, 1995), and references therein.

### 2.3. The VC Dimension and the Number of Parameters

The VC dimension thus gives concreteness to the notion of the capacity of a given set of functions. Intuitively, one might be led to expect that learning machines with many parameters would have high VC dimension, while learning machines with few parameters would have low VC dimension. There is a striking counterexample to this, due to E. Levin and J.S. Denker (Vapnik, 1995): A learning machine with just one parameter, but with infinite VC dimension (a family of classifiers is said to have infinite VC dimension if it can shatter $l$ points, no matter how large $l$). Define the step function $\theta(x)$, $x \in \mathbb{R}$: $\{\theta(x) = 1 \forall x > 0; \quad \theta(x) = -1 \forall x \leq 0\}$. Consider the one-parameter family of functions, defined by

$$f(x, \alpha) \equiv \theta(\sin(\alpha x)), \quad x, \alpha \in \mathbb{R}. \quad (4)$$

You choose some number $l$, and present me with the task of finding $l$ points that can be shattered. I choose them to be:

$$x_i = 10^{-i}, \quad i = 1, \cdots, l. \quad (5)$$

You specify any labels you like:

$$y_1, y_2, \cdots, y_l, \quad y_i \in \{-1, 1\}. \quad (6)$$

Then $f(\alpha)$ gives this labeling if I choose $\alpha$ to be

$$\alpha = \pi(1 + \sum_{i=1}^{l} \frac{(1 - y_i)10^i}{2}). \quad (7)$$

Thus the VC dimension of this machine is infinite.

Interestingly, even though we can shatter an arbitrarily large number of points, we can also find just four points that cannot be shattered. They simply have to be equally spaced, and assigned labels as shown in Figure 2. This can be seen as follows: Write the phase at $x_1$ as $\phi_1 = 2\pi x + \delta$. Then the choice of label $y_1 = 1$ requires $0 < \delta < \pi$. The phase at $x_2$, mod $2\pi$, is $2\delta$; then $y_2 = 1 \Rightarrow 0 < \delta < \pi/2$. Similarly, point $x_3$ forces $\delta > \pi/3$. Then at $x_4$, $\pi/3 < \delta < \pi/2$ implies that $f(x_4, \alpha) = -1$, contrary to the assigned label. These four points are the analogy, for the set of functions in Eq. (4), of the set of three points lying along a line, for oriented hyperplanes in $\mathbb{R}^n$. Neither set can be shattered by the chosen family of functions.
Figure 2. Four points that cannot be shattered by $\theta(\sin(ax))$, despite infinite VC dimension.

\[ \begin{array}{c|cccc} x = 0 & 1 & 2 & 3 & 4 \\ \hline \end{array} \]

\[ \begin{array}{c|cccc} 0 & 0.1 & 0.2 & 0.3 & 0.4 \\ \hline \end{array} \]

\[ \begin{array}{c|cccc} 0.5 & 0.6 & 0.7 & 0.8 & 0.9 \\ \hline \end{array} \]

\[ \begin{array}{c|cccc} 1 & 1.2 & 1.3 & 1.4 \end{array} \]

Figure 3. VC confidence is monotonic in $h$

2.4. Minimizing The Bound by Minimizing $h$

Figure 3 shows how the second term on the right hand side of Eq. (3) varies with $h$, given a choice of 95% confidence level ($\eta = 0.05$) and assuming a training sample of size 10,000. The VC confidence is a monotonic increasing function of $h$. This will be true for any value of $l$.

Thus, given some selection of learning machines whose empirical risk is zero, one wants to choose that learning machine whose associated set of functions has minimal VC dimension. This will lead to a better upper bound on the actual error. In general, for non zero empirical risk, one wants to choose that learning machine which minimizes the right hand side of Eq. (3).

Note that in adopting this strategy, we are only using Eq. (3) as a guide. Eq. (3) gives (with some chosen probability) an upper bound on the actual risk. This does not prevent a particular machine with the same value for empirical risk, and whose function set has higher VC dimension, from having better performance. In fact an example of a system that gives good performance despite having infinite VC dimension is given in the next Section. Note also that the graph shows that for $h/l > 0.37$ (and for $\eta = 0.05$ and $l = 10,000$), the VC confidence exceeds unity, and so for higher values the bound is guaranteed not tight.

2.5. Two Examples

Consider the $k$'th nearest neighbour classifier, with $k = 1$. This set of functions has infinite VC dimension and zero empirical risk, since any number of points, labeled arbitrarily, will be successfully learned by the algorithm (provided no two points of opposite class are right on top of each other). Thus the bound provides no information. In fact, for any classifier with infinite VC dimension, the bound is not even valid. However, even though the bound
is not valid, nearest neighbour classifiers can still perform well. Thus this first example is a cautionary tale: infinite “capacity” does not guarantee poor performance.

Let’s follow the time honoured tradition of understanding things by trying to break them, and see if we can come up with a classifier for which the bound is supposed to hold, but which violates the bound. We want the left hand side of Eq. (3) to be as large as possible, and the right hand side to be as small as possible. So we want a family of classifiers which gives the worst possible actual risk of 0.5, zero empirical risk up to some number of training observations, and whose VC dimension is easy to compute and is less than \( l \) (so that the bound is non trivial). An example is the following, which I call the “notebook classifier.”

This classifier consists of a notebook with enough room to write down the classes of \( m \) training observations, where \( m \leq l \). For all subsequent patterns, the classifier simply says that all patterns have the same class. Suppose also that the data have as many positive \((y = +1)\) as negative \((y = -1)\) examples, and that the samples are chosen randomly. The notebook classifier will have zero empirical risk for up to \( m \) observations; 0.5 training error for all subsequent observations; 0.5 actual error, and VC dimension \( h = m \). Substituting these values in Eq. (3), the bound becomes:

\[
\frac{m}{4l} \leq \ln(2l/m) + 1 - (1/m) \ln(\eta/4)
\]

which is certainly met for all \( \eta \) if

\[
f(z) = \left(\frac{z}{2}\right) \exp\left(\frac{z}{4}-1\right) \leq 1, \quad z \equiv (m/l), \quad 0 \leq z \leq 1
\]

which is true, since \( f(z) \) is monotonic increasing, and \( f(z = 1) = 0.236 \).

2.6. Structural Risk Minimization

We can now summarize the principle of structural risk minimization (SRM) (Vapnik, 1979). Note that the VC confidence term in Eq. (3) depends on the chosen class of functions, whereas the empirical risk and actual risk depend on the one particular function chosen by the training procedure. We would like to find that subset of the chosen set of functions, such that the risk bound for that subset is minimized. Clearly we cannot arrange things so that the VC dimension \( h \) varies smoothly, since it is an integer. Instead, introduce a “structure” by dividing the entire class of functions into nested subsets (Figure 4). For each subset, we must be able either to compute \( h \), or to get a bound on \( h \) itself. SRM then consists of finding that subset of functions which minimizes the bound on the actual risk. This can be done by simply training a series of machines, one for each subset, where for a given subset the goal of training is simply to minimize the empirical risk. One then takes that trained machine in the series whose sum of empirical risk and VC confidence is minimal.

![Figure 4. Nested subsets of functions, ordered by VC dimension.](image)
We have now laid the groundwork necessary to begin our exploration of support vector machines.

3. Linear Support Vector Machines

3.1. The Separable Case

We will start with the simplest case: linear machines trained on separable data (as we shall see, the analysis for the general case - nonlinear machines trained on non-separable data - results in a very similar quadratic programming problem). Again label the training data \( \{x_i, y_i\}, \ i = 1, \ldots, l, \ y_i \in \{-1, 1\}, \ x_i \in \mathbb{R}^d. \) Suppose we have some hyperplane which separates the positive from the negative examples (a "separating hyperplane"). The points \( x \) which lie on the hyperplane satisfy \( w \cdot x + b = 0, \) where \( w \) is normal to the hyperplane, \( |b|/||w|| \) is the perpendicular distance from the hyperplane to the origin, and \( ||w|| \) is the Euclidean norm of \( w. \) Let \( d_+ (d_-) \) be the shortest distance from the separating hyperplane to the closest positive (negative) example. Define the "margin" of a separating hyperplane to be \( d_+ + d_- \). For the linearly separable case, the support vector algorithm simply looks for the separating hyperplane with largest margin. This can be formulated as follows: suppose that all the training data satisfy the following constraints:

\[
\begin{align*}
\mathbf{x}_i \cdot \mathbf{w} + b & \geq +1 \quad \text{for} \ y_i = +1 \quad (10) \\
\mathbf{x}_i \cdot \mathbf{w} + b & \leq -1 \quad \text{for} \ y_i = -1 \quad (11)
\end{align*}
\]

These can be combined into one set of inequalities:

\[
y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \geq 0 \quad \forall i \quad (12)
\]

Now consider the points for which the equality in Eq. (10) holds (requiring that there exists such a point is equivalent to choosing a scale for \( w \) and \( b \)). These points lie on the hyperplane \( H_1 : \mathbf{x}_i \cdot \mathbf{w} + b = 1 \) with normal \( w \) and perpendicular distance from the origin \( |1 - b|/||w||. \) Similarly, the points for which the equality in Eq. (11) holds lie on the hyperplane \( H_2 : \mathbf{x}_i \cdot \mathbf{w} + b = -1 \), with normal again \( w \), and perpendicular distance from the origin \( |1 - b|/||w||. \) Hence \( d_+ = d_- = 1/||w|| \) and the margin is simply \( 2/||w||. \) Note that \( H_1 \) and \( H_2 \) are parallel (they have the same normal) and that no training points fall between them. Thus we can find the pair of hyperplanes which gives the maximum margin by minimizing \( ||w||^2, \) subject to constraints (12).

Thus we expect the solution for a typical two dimensional case to have the form shown in Figure 5. Those training points for which the equality in Eq. (12) holds (i.e. those which wind up lying on one of the hyperplanes \( H_1, H_2 \), and whose removal would change the solution found, are called support vectors; they are indicated in Figure 5 by the extra circles.

We will now switch to a Lagrangian formulation of the problem. There are two reasons for doing this. The first is that the constraints (12) will be replaced by constraints on the Lagrange multipliers themselves, which will be much easier to handle. The second is that in this reformulation of the problem, the training data will only appear (in the actual training and test algorithms) in the form of dot products between vectors. This is a crucial property which will allow us to generalize the procedure to the nonlinear case (Section 4).

Thus, we introduce positive Lagrange multipliers \( \alpha_i, \ i = 1, \ldots, l, \) one for each of the inequality constraints (12). Recall that the rule is that for constraints of the form \( c_i \geq 0, \)
the constraint equations are multiplied by positive Lagrange multipliers and subtracted from the objective function, to form the Lagrangian. For equality constraints, the Lagrange multipliers are unconstrained. This gives Lagrangian:

\[ L_P \equiv \frac{1}{2} \|w\|^2 - \sum_{i=1}^{l} \alpha_i y_i (x_i \cdot w + b) + \sum_{i=1}^{l} \alpha_i \]  \hspace{1cm} (13)

We must now minimize \( L_P \) with respect to \( w \), \( b \), and simultaneously require that the derivatives of \( L_P \) with respect to all the \( \alpha_i \) vanish, all subject to the constraints \( \alpha_i \geq 0 \) (let’s call this particular set of constraints \( C_1 \)). Now this is a convex quadratic programming problem, since the objective function is itself convex, and those points which satisfy the constraints also form a convex set (any linear constraint defines a convex set, and a set of \( N \) simultaneous linear constraints defines the intersection of \( N \) convex sets, which is also a convex set). This means that we can equivalently solve the following “dual” problem: maximize \( L_P \), subject to the constraints that the gradient of \( L_P \) with respect to \( w \) and \( b \) vanish, and subject also to the constraints that the \( \alpha_i \geq 0 \) (let’s call that particular set of constraints \( C_2 \)). This particular dual formulation of the problem is called the Wolfe dual (Fletcher, 1987). It has the property that the maximum of \( L_P \), subject to constraints \( C_2 \), occurs at the same values of the \( w \), \( b \) and \( \alpha \), as the minimum of \( L_P \), subject to constraints \( C_1 \).

Requiring that the gradient of \( L_P \) with respect to \( w \) and \( b \) vanish give the conditions:

\[ w = \sum_i \alpha_i y_i x_i \]  \hspace{1cm} (14)

\[ \sum_i \alpha_i y_i = 0. \]  \hspace{1cm} (15)

Since these are equality constraints in the dual formulation, we can substitute them into Eq. (13) to give

\[ L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i \cdot x_j \]  \hspace{1cm} (16)

Figure 5. Linear separating hyperplanes for the separable case. The support vectors are circled.
Note that we have now given the Lagrangian different labels \( P \) for primal, \( D \) for dual to emphasize that the two formulations are different: \( L_P \) and \( L_D \) arise from the same objective function but with different constraints; and the solution is found by minimizing \( L_P \) or by maximizing \( L_D \). Note also that if we formulate the problem with \( b = 0 \), which amounts to requiring that all hyperplanes contain the origin, the constraint (15) does not appear. This is a mild restriction for high dimensional spaces, since it amounts to reducing the number of degrees of freedom by one.

Support vector training (for the separable, linear case) therefore amounts to maximizing \( L_D \) with respect to the \( \alpha_i \), subject to constraints (15) and positivity of the \( \alpha_i \), with solution given by (14). Notice that there is a Lagrange multiplier \( \alpha_i \) for every training point. In the solution, those points for which \( \alpha_i > 0 \) are called “support vectors”, and lie on one of the hyperplanes \( H_1, H_2 \). All other training points have \( \alpha_i = 0 \) and lie either on \( H_1 \) or \( H_2 \) (such that the equality in Eq. (12) holds), or on that side of \( H_1 \) or \( H_2 \) such that the strict inequality in Eq. (12) holds. For these machines, the support vectors are the critical elements of the training set. They lie closest to the decision boundary; if all other training points were removed (or moved around, but so as not to cross \( H_1 \) or \( H_2 \)), and training was repeated, the same separating hyperplane would be found.

3.2. The Karush-Kuhn-Tucker Conditions

The Karush-Kuhn-Tucker (KKT) conditions play a central role in both the theory and practice of constrained optimization. For the primal problem above, the KKT conditions may be stated (Fletcher, 1987):

\[
\frac{\partial}{\partial \omega} L_P = w_{\nu} - \sum_i \alpha_i y_i x_{i \nu} = 0 \quad \nu = 1, \ldots, d
\]  
\[
\frac{\partial}{\partial b} L_P = - \sum_i \alpha_i y_i = 0
\]  
\[
y_i (x_i \cdot w + b) - 1 \geq 0 \quad i = 1, \ldots, l
\]  
\[
\alpha_i \geq 0 \quad \forall i
\]  
\[
\alpha_i (y_i (w \cdot x_i + b) - 1) = 0 \quad \forall i
\]  

The KKT conditions are satisfied at the solution of any constrained optimization problem (convex or not), with any kind of constraints, provided that the intersection of the set of feasible directions with the set of descent directions coincides with the intersection of the set of feasible directions for linearized constraints with the set of descent directions (see Fletcher, 1987; McCormick, 1983). This rather technical regularity assumption holds for all support vector machines, since the constraints are always linear. Furthermore, the problem for SVMs is convex (a convex objective function, with constraints which give a convex feasible region), and for convex problems (if the regularity condition holds), the KKT conditions are necessary and sufficient for \( w, b, \alpha \) to be a solution (Fletcher, 1987). Thus solving the SVM problem is equivalent to finding a solution to the KKT conditions. This fact results in several approaches to finding the solution (for example, the primal-dual path following method mentioned in Section 5).

As an immediate application, note that, while \( w \) is explicitly determined by the training procedure, the threshold \( b \) is not, although it is implicitly determined. However \( b \) is easily found by using the KKT “complementarity” condition, Eq. (21), by choosing any \( i \) for
which $\alpha_i \neq 0$ and computing $b$ (note that it is numerically safer to take the mean value of $b$ resulting from all such equations).

Notice that all we've done so far is to cast the problem into an optimization problem where the constraints are rather more manageable than those in Eqs. (10), (11). Finding the solution for real world problems will usually require numerical methods. We will have more to say on this later. However, let's first work out a rare case where the problem is nontrivial (the number of dimensions is arbitrary, and the solution certainly not obvious), but where the solution can be found analytically.

3.3. Optimal Hyperplanes: An Example

While the main aim of this Section is to explore a non-trivial pattern recognition problem where the support vector solution can be found analytically, the results derived here will also be useful in a later proof. For the problem considered, every training point will turn out to be a support vector, which is one reason we can find the solution analytically.

Consider $n + 1$ symmetrically placed points lying on a sphere $S^{n-1}$ of radius $R$: more precisely, the points form the vertices of an $n$-dimensional symmetric simplex. It is convenient to embed the points in $\mathbb{R}^{n+1}$ in such a way that they all lie in the hyperplane which passes through the origin and which is perpendicular to the $(n+1)$-vector $(1, 1, ..., 1)$ (in this formulation, the points lie on $S^{n-1}$, they span $\mathbb{R}^n$, and are embedded in $\mathbb{R}^{n+1}$). Explicitly, recalling that vectors themselves are labeled by Roman indices and their coordinates by Greek, the coordinates are given by:

$$x_{i\mu} = -(1 - \delta_{i\mu}) \sqrt{\frac{R}{n(n + 1)}} + \delta_{i\mu} \sqrt{\frac{Rn}{n + 1}}$$  \hspace{1cm} (22)

where the Kronecker delta, $\delta_{i\mu}$, is defined by $\delta_{i\mu} = 1$ if $\mu = i$, 0 otherwise. Thus, for example, the vectors for three equidistant points on the unit circle (see Figure 12) are:

$$x_1 = \left(\sqrt{\frac{2}{3}}, \frac{-1}{\sqrt{6}}, \frac{-1}{\sqrt{6}}\right)$$

$$x_2 = \left(\frac{-1}{\sqrt{6}}, \sqrt{\frac{2}{3}}, \frac{-1}{\sqrt{6}}\right)$$

$$x_3 = \left(\frac{-1}{\sqrt{6}}, \frac{-1}{\sqrt{6}}, \sqrt{\frac{2}{3}}\right)$$  \hspace{1cm} (23)

One consequence of the symmetry is that the angle between any pair of vectors is the same (and is equal to $\arccos(-1/n)$):

$$||x_i||^2 = R^2$$  \hspace{1cm} (24)

$$x_i \cdot x_j = -\frac{R^2}{n}$$  \hspace{1cm} (25)

or, more succinctly,

$$\frac{x_i \cdot x_j}{R^2} = \delta_{i,j} - (1 - \delta_{i,j}) \frac{1}{n}.$$  \hspace{1cm} (26)

Assigning a class label $C \in \{+1,-1\}$ arbitrarily to each point, we wish to find that hyperplane which separates the two classes with widest margin. Thus we must maximize
\( L_D \) in Eq. (16), subject to \( \alpha_i \geq 0 \) and also subject to the equality constraint, Eq. (15). Our strategy is to simply solve the problem as though there were no inequality constraints. If the resulting solution does in fact satisfy \( \alpha_i \geq 0 \forall i \), then we will have found the general solution, since the actual maximum of \( L_D \) will then lie in the feasible region, provided the equality constraint, Eq. (15), is also met. In order to impose the equality constraint we introduce an additional Lagrange multiplier \( \lambda \). Thus we seek to maximize

\[
L_D = \sum_{i=1}^{n+1} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n+1} \alpha_i \alpha_j \lambda - \sum_{i=1}^{n+1} \alpha_i y_i, \tag{27}
\]

where we have introduced the Hessian

\[
H_{ij} = y_i y_j x_i \cdot x_j. \tag{28}
\]

Setting \( \frac{\partial L_D}{\partial \alpha_i} = 0 \) gives

\[
(H \alpha)_i + \lambda y_i = 1 \quad \forall i \tag{29}
\]

Now \( H \) has a very simple structure: the off-diagonal elements are \( -y_i y_j R^2 / n \), and the diagonal elements are \( R^2 \). The fact that all the off-diagonal elements differ only by factors of \( y_i \) suggests looking for a solution which has the form:

\[
\alpha_i = \left( 1 + \frac{y_i}{2} \right) a + \left( \frac{1 - y_i}{2} \right) b \tag{30}
\]

where \( a \) and \( b \) are unknowns. Plugging this form in Eq. (29) gives:

\[
\left( \frac{n+1}{n} \right) \left( \frac{a + b}{2} \right) - \frac{y_i}{n} \left( \frac{a + b}{2} \right) = \frac{1 - \lambda y_i}{R^2} \tag{31}
\]

where \( p \) is defined by

\[
p = \sum_{i=1}^{n+1} y_i. \tag{32}
\]

Thus

\[
a + b = \frac{2n}{R^2(n+1)} \tag{33}
\]

and substituting this into the equality constraint Eq. (15) to find \( a, b \) gives

\[
a = \frac{n}{R^2(n+1)} \left( 1 - \frac{p}{n+1} \right), \quad b = \frac{n}{R^2(n+1)} \left( 1 + \frac{p}{n+1} \right) \tag{34}
\]

which gives for the solution

\[
\alpha_i = \frac{n}{R^2(n+1)} \left( 1 - \frac{y_i p}{n+1} \right) \tag{35}
\]

Also,

\[
(H \alpha)_i = 1 - \frac{y_i p}{n+1}. \tag{36}
\]
Hence

$$\|w\|^2 = \sum_{i,j=1}^{n+1} \alpha_i \alpha_j y_i y_j x_i \cdot x_j = \alpha^T H \alpha$$

$$= \sum_{i=1}^{n+1} \alpha_i \left(1 - \frac{y_i p}{n+1}\right) = \sum_{i=1}^{n+1} \alpha_i \left(\frac{n}{R^2} \left(1 - \left(\frac{p}{n+1}\right)^2\right)\right)$$  \hspace{1cm} (37)

Note that this is one of those cases where the Lagrange multiplier $\lambda$ can remain undetermined (although determining it is trivial). We have now solved the problem, since all the $\alpha_i$ are clearly positive or zero (in fact the $\alpha_i$ will only be zero if all training points have the same class). Note that $\|w\|$ depends only on the number of positive (negative) polarity points, and not on how the class labels are assigned to the points in Eq. (22). This is clearly not true of $w$ itself, which is given by

$$w = \frac{n}{R^2(n+1)} \sum_{i=1}^{n+1} \left(y_i - \frac{p}{n+1}\right) x_i$$  \hspace{1cm} (38)

The margin, $M = 2/\|w\|$, is thus given by

$$M = \frac{2R}{\sqrt{n(1 - (p/(n+1))^2)}}.$$  \hspace{1cm} (39)

Thus when the number of points $n+1$ is even, the minimum margin occurs when $p = 0$ (equal numbers of positive and negative examples), in which case the margin is $M_{\text{min}} = 2R/\sqrt{n}$. If $n+1$ is odd, the minimum margin occurs when $p = \pm 1$, in which case $M_{\text{min}} = 2R(n+1)/(n\sqrt{n+2})$. In both cases, the maximum margin is given by $M_{\text{max}} = R(n+1)/n$. Thus, for example, for the two dimensional simplex consisting of three points lying on $\mathbb{S}^1$ (and spanning $\mathbb{R}^2$), and with labeling such that not all three points have the same polarity, the maximum and minimum margin are both $3R/2$ (see Figure (12)).

Note that the results of this Section amount to an alternative, constructive proof that the VC dimension of oriented separating hyperplanes in $\mathbb{R}^3$ is at least $n+1$.

3.4. Test Phase

Once we have trained a Support Vector Machine, how can we use it? We simply determine on which side of the decision boundary (that hyperplane lying half way between $H_1$ and $H_2$ and parallel to them) a given test pattern $x$ lies and assign the corresponding class label, i.e. we take the class of $x$ to be $\text{sgn}(w \cdot x + b)$.

3.5. The Non-Separable Case

The above algorithm for separable data, when applied to non-separable data, will find no feasible solution: this will be evidenced by the objective function (i.e. the dual Lagrangian) growing arbitrarily large. So how can we extend these ideas to handle non-separable data? We would like to relax the constraints (10) and (11), but only when necessary, that is, we would like to introduce a further cost (i.e. an increase in the primal objective function) for doing so. This can be done by introducing positive slack variables $\xi_i$, $i = 1, \cdots, l$ in the constraints (Cortes and Vapnik, 1995), which then become:
\[ \mathbf{x}_i \cdot \mathbf{w} + b \geq +1 - \xi_i \quad \text{for } y_i = +1 \]  
\[ \mathbf{x}_i \cdot \mathbf{w} + b \leq -1 + \xi_i \quad \text{for } y_i = -1 \]  
\[ \xi_i \geq 0 \quad \forall i. \]  

Thus, for an error to occur, the corresponding \( \xi_i \) must exceed unity, so \( \sum_i \xi_i \) is an upper bound on the number of training errors. Hence a natural way to assign an extra cost for errors is to change the objective function to be minimized from \( \| \mathbf{w} \|^2 / 2 \) to \( \| \mathbf{w} \|^2 / 2 + C (\sum_i \xi_i)^k \), where \( C \) is a parameter to be chosen by the user, a larger \( C \) corresponding to assigning a higher penalty to errors. As it stands, this is a convex programming problem for any positive integer \( k \); for \( k = 2 \) and \( k = 1 \) it is also a quadratic programming problem, and the choice \( k = 1 \) has the further advantage that neither the \( \xi_i \) nor their Lagrange multipliers, appear in the Wolfe dual problem, which becomes:

**Maximize:**

\[ L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \]  

**subject to:**

\[ 0 \leq \alpha_i \leq C, \]  
\[ \sum_i \alpha_i y_i = 0. \]

The solution is again given by

\[ \mathbf{w} = \sum_{i=1}^{N_S} \alpha_i y_i \mathbf{x}_i. \]  

where \( N_S \) is the number of support vectors. Thus the only difference from the optimal hyperplane case is that the \( \alpha_i \) now have an upper bound of \( C \). The situation is summarized schematically in Figure 6.

We will need the Karush-Kuhn-Tucker conditions for the primal problem. The primal Lagrangian is

\[ L_P = \frac{1}{2} \| \mathbf{w} \|^2 + C \sum_i \xi_i - \sum_i \alpha_i \{ y_i (\mathbf{x}_i \cdot \mathbf{w} + b) - 1 + \xi_i \} - \sum_i \mu_i \xi_i \]  

where the \( \mu_i \) are the Lagrange multipliers introduced to enforce positivity of the \( \xi_i \). The KKT conditions for the primal problem are therefore (note \( i \) runs from 1 to the number of training points, and \( \nu \) from 1 to the dimension of the data)

\[ \frac{\partial L_P}{\partial w_{i\nu}} = w_{i\nu} - \sum_i \alpha_i y_i x_{i\nu} = 0 \]  
\[ \frac{\partial L_P}{\partial b} = -\sum_i \alpha_i y_i = 0 \]
\[
\frac{\partial L_P}{\partial \xi_i} = C - \alpha_i - \mu_i = 0 \quad (50)
\]
\[
y_i(x_i \cdot w + b) - 1 + \xi_i \geq 0 \quad (51)
\]
\[
\xi_i \geq 0 \quad (52)
\]
\[
\alpha_i \geq 0 \quad (53)
\]
\[
\mu_i \geq 0 \quad (54)
\]
\[
\alpha_i \{y_i(x_i \cdot w + b) - 1 + \xi_i\} = 0 \quad (55)
\]
\[
\mu_i \xi_i = 0 \quad (56)
\]

As before, we can use the KKT complementarity conditions, Eqs. (55) and (56), to determine the threshold \(b\). Note that Eq. (50) combined with Eq. (56) shows that \(\xi_i = 0\) if \(\alpha_i < C\). Thus we can simply take any training point for which \(0 < \alpha_i < C\) to use Eq. (55) (with \(\xi_i = 0\)) to compute \(b\). (As before, it is numerically wiser to take the average over all such training points.)

\[
\sum_{i} \text{Forces} = \sum_{i} \alpha_i y_i \hat{w} = 0 \quad (57)
\]
\[
\sum_{i} \text{Torques} = \sum_{i} s_i \wedge (\alpha_i y_i \hat{w}) = \hat{w} \wedge w = 0. \quad (58)
\]

(Here the \(s_i\) are the support vectors, and \(\wedge\) denotes the vector product.) For data in \(\mathbb{R}^n\), clearly the condition that the sum of forces vanish is still met. One can easily show that the torque also vanishes.\(^9\)

This mechanical analogy depends only on the form of the solution (46), and therefore holds for both the separable and the non-separable cases. In fact this analogy holds in general
(i.e., also for the nonlinear case described below). The analogy emphasizes the interesting point that the “most important” data points are the support vectors with highest values of \( \alpha_i \), since they exert the highest forces on the decision sheet. For the non-separable case, the upper bound \( \alpha_i \leq C \) corresponds to an upper bound on the force any given point is allowed to exert on the sheet. This analogy also provides a reason (as good as any other) to call these particular vectors “support vectors”.

### 3.7. Examples by Pictures

Figure 7 shows two examples of a two-class pattern recognition problem, one separable and one not. The two classes are denoted by circles and disks respectively. Support vectors are identified with an extra circle. The error in the non-separable case is identified with a cross. The reader is invited to use Lucent’s SVM Applet (Burges, Knirsch and Haratsch, 1996) to experiment and create pictures like these (if possible, try using 16 or 24 bit color).

![Figure 7](image1.png)  
![Figure 7](image2.png)

*Figure 7*: The linear case, separable (left) and not (right). The background colour shows the shape of the decision surface.

### 4. Nonlinear Support Vector Machines

How can the above methods be generalized to the case where the decision function is not a linear function of the data? (Boser, Guyon and Vapnik, 1992), showed that a rather old trick (Aizerman, 1964) can be used to accomplish this in an astonishingly straightforward way. First notice that the only way in which the data appears in the training problem, Eqs. (43) - (45), is in the form of dot products, \( \mathbf{x}_i \cdot \mathbf{x}_j \). Now suppose we first mapped the data to some other (possibly infinite dimensional) Euclidean space \( \mathcal{H} \), using a mapping which we will call \( \Phi \):

\[
\Phi : \mathbb{R}^d \rightarrow \mathcal{H}.
\]  

Then of course the training algorithm would only depend on the data through dot products in \( \mathcal{H} \), i.e. on functions of the form \( \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) \). Now if there were a “kernel function” \( K \) such that \( K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) \), we would only need to use \( K \) in the training algorithm, and would never need to explicitly even know what \( \Phi \) is. One example is

\[
\Phi(\mathbf{x}) = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \\ x_1 x_2 \\ x_1 x_3 \\ \vdots \\ x_1 x_{d-1} \\ x_2 x_3 \\ \vdots \\ x_2 x_{d-1} \\ \vdots \\ x_{d-1} x_d \end{bmatrix}
\]
\[ K(x_i, x_j) = e^{-\|x_i - x_j\|^2 / 2\sigma^2}. \] (60)

In this particular example, \( \mathcal{H} \) is infinite dimensional, so it would not be very easy to work with \( \Phi \) explicitly. However, if one replaces \( x_i \cdot x_j \) by \( K(x_i, x_j) \) everywhere in the training algorithm, the algorithm will happily produce a support vector machine which lives in an infinite dimensional space, and furthermore do so in roughly the same amount of time it would take to train on the un-mapped data. All the considerations of the previous sections hold, since we are still doing a linear separation, but in a different space.

But how can we use this machine? After all, we need \( \mathbf{w} \), and that will live in \( \mathcal{H} \) also (see Eq. (46)). But in test phase an SVM is used by computing dot products of a given test point \( x \) with \( \mathbf{w} \), or more specifically by computing the sign of

\[ f(x) = \sum_{i=1}^{N_S} \alpha_i y_i \Phi(s_i) \cdot \Phi(x) + b = \sum_{i=1}^{N_S} \alpha_i y_i K(s_i, x) + b \] (61)

where the \( s_i \) are the support vectors. So again we can avoid computing \( \Phi(x) \) explicitly and use the \( K(s_i, x) = \Phi(s_i) \cdot \Phi(x) \) instead.

Let us call the space in which the data live, \( \mathcal{L} \). (Here and below we use \( \mathcal{L} \) as a mnemonic for “low dimensional”, and \( \mathcal{H} \) for “high dimensional”: it is usually the case that the range of \( \Phi \) is of much higher dimension than its domain.) Note that, in addition to the fact that \( \mathbf{w} \) lives in \( \mathcal{H} \), there will in general be no vector in \( \mathcal{L} \) which maps, via the map \( \Phi \), to \( \mathbf{w} \). If there were, \( f(x) \) in Eq. (61) could be computed in one step, avoiding the sum (and making the corresponding SVM \( N_S \) times faster, where \( N_S \) is the number of support vectors). Despite this, ideas along these lines can be used to significantly speed up the test phase of SVMs (Burges, 1996). Note also that it is easy to find kernels (for example, kernels which are functions of the dot products of the \( x_i \) in \( \mathcal{L} \)) such that the training algorithm and solution found are independent of the dimension of both \( \mathcal{L} \) and \( \mathcal{H} \).

In the next Section we will discuss which functions \( K \) are allowable and which are not. Let us end this Section with a very simple example of an allowed kernel, for which we can construct the mapping \( \Phi \).

Suppose that your data are vectors in \( \mathbb{R}^2 \), and you choose \( K(x_i, x_j) = (x_i \cdot x_j)^2 \). Then it’s easy to find a space \( \mathcal{H} \), and mapping \( \Phi \) from \( \mathbb{R}^2 \) to \( \mathcal{H} \), such that \( (x \cdot y)^2 = \Phi(x) \cdot \Phi(y) \): we choose \( \mathcal{H} = \mathbb{R}^3 \) and

\[ \Phi(x) = \begin{pmatrix} x_1^2 \\ \sqrt{2} x_1 x_2 \\ x_2^2 \end{pmatrix} \] (62)

(note that here the subscripts refer to vector components). For data in \( \mathcal{L} \) defined on the square \([-1, 1] \times [-1, 1] \in \mathbb{R}^2 \) (a typical situation, for grey level image data), the (entire) image of \( \Phi \) is shown in Figure 8. This Figure also illustrates how to think of this mapping: the image of \( \Phi \) may live in a space of very high dimension, but it is just a (possibly very contorted) surface whose intrinsic dimension\(^{12} \) is just that of \( \mathcal{L} \).

Note that neither the mapping \( \Phi \) nor the space \( \mathcal{H} \) are unique for a given kernel. We could equally well have chosen \( \mathcal{H} \) to again be \( \mathbb{R}^3 \) and

\[ \Phi(x) = \frac{1}{\sqrt{2}} \begin{pmatrix} (x_1^2 - x_2^2) \\ 2x_1 x_2 \\ (x_1^2 + x_2^2) \end{pmatrix} \] (63)
or \( \mathcal{H} \) to be \( \mathbb{R}^4 \) and

\[
\Phi(x) = \begin{pmatrix}
  x_1^2 \\
  x_1 x_2 \\
  x_1 x_2 \\
  x_2^2
\end{pmatrix}.
\]

(64)

The literature on SVMs usually refers to the space \( \mathcal{H} \) as a Hilbert space, so let’s end this Section with a few notes on this point. You can think of a Hilbert space as a generalization of Euclidean space that behaves in a gentlemanly fashion. Specifically, it is any linear space, with an inner product defined, which is also complete with respect to the corresponding norm (that is, any Cauchy sequence of points converges to a point in the space). Some authors (e.g. (Kolmogorov, 1970)) also require that it be separable (that is, it must have a countable subset whose closure is the space itself), and some (e.g. Halmos, 1967) don’t. It’s a generalization mainly because its inner product can be any inner product, not just the scalar (“dot”) product used here (and in Euclidean spaces in general). It’s interesting that the older mathematical literature (e.g. Kolmogorov, 1970) also required that Hilbert spaces be infinite dimensional, and that mathematicians are quite happy defining infinite dimensional Euclidean spaces. Research on Hilbert spaces centers on operators in those spaces, since the basic properties have long since been worked out. Since some people understandably blanch at the mention of Hilbert spaces, I decided to use the term Euclidean throughout this tutorial.

4.1. Mercer’s Condition

For which kernels does there exist a pair \( \{ \mathcal{H}, \Phi \} \), with the properties described above, and for which does there not? The answer is given by Mercer’s condition (Vapnik, 1995; Courant and Hilbert, 1953): There exists a mapping \( \Phi \) and an expansion

\[
K(x,y) = \sum_i \Phi(x)_i \Phi(y)_i
\]

(65)
if and only if, for any \( g(x) \) such that
\[
\int g(x)^2 \, dx \quad \text{is finite}
\] (66) then
\[
\int K(x, y)g(x)g(y) \, dx \, dy \geq 0.
\] (67)

Note that for specific cases, it may not be easy to check whether Mercer's condition is satisfied. Eq. (67) must hold for every \( g \) with finite \( L_2 \) norm (i.e. which satisfies Eq. (66)). However, we can easily prove that the condition is satisfied for positive integral powers of the dot product: \( K(x, y) = (x \cdot y)^p \). We must show that
\[
\int \left( \sum_{i=1}^{d} x_i y_i \right)^p g(x)g(y) \, dx \, dy \geq 0.
\] (68)

The typical term in the multinomial expansion of \( \left( \sum_{i=1}^{d} x_i y_i \right)^p \) contributes a term of the form
\[
\frac{p!}{r_1! r_2! \cdots (p - r_1 - r_2 \cdots)!} \int x_1^{r_1} x_2^{r_2} \cdots y_1^{r_1} y_2^{r_2} \cdots g(x)g(y) \, dx \, dy
\] (69)
to the left hand side of Eq. (67), which factorizes:
\[
= \frac{p!}{r_1! r_2! \cdots (p - r_1 - r_2 \cdots)!} \left( \int x_1^{r_1} x_2^{r_2} \cdots g(x)^2 \right) \geq 0.
\] (70)

One simple consequence is that any kernel which can be expressed as \( K(x, y) = \sum_{p=0}^{\infty} c_p (x \cdot y)^p \), where the \( c_p \) are positive real coefficients and the series is uniformly convergent, satisfies Mercer's condition, a fact also noted in (Smola, Schölkopf and Müller, 1998b).

Finally, what happens if one uses a kernel which does not satisfy Mercer's condition? In general, there may exist data such that the Hessian is indefinite, and for which the quadratic programming problem will have no solution (the dual objective function can become arbitrarily large). However, even for kernels that do not satisfy Mercer's condition, one might still find that a given training set results in a positive semidefinite Hessian, in which case the training will converge perfectly well. In this case, however, the geometrical interpretation described above is lacking.

4.2. **Some Notes on \( \Phi \) and \( \mathcal{H} \)**

Mercer's condition tells us whether or not a prospective kernel is actually a dot product in some space, but it does not tell us how to construct \( \Phi \) or even what \( \mathcal{H} \) is. However, as with the homogeneous (that is, homogeneous in the dot product in \( \mathcal{L} \)) quadratic polynomial kernel discussed above, we can explicitly construct the mapping for some kernels. In Section 6.1 we show how Eq. (62) can be extended to arbitrary homogeneous polynomial kernels, and that the corresponding space \( \mathcal{H} \) is a Euclidean space of dimension \((d+\frac{p-1}{p})\). Thus for example, for a degree \( p = 4 \) polynomial, and for data consisting of 16 by 16 images (\( d=256 \)), \( \dim(\mathcal{H}) \) is 183,181,376.

Usually, mapping your data to a “feature space” with an enormous number of dimensions would bode ill for the generalization performance of the resulting machine. After all, the
set of all hyperplanes \{w, b\} are parameterized by \(\dim(\mathcal{H}) + 1\) numbers. Most pattern recognition systems with billions, or even an infinite, number of parameters would not make it past the start gate. How come SVMs do so well? One might argue that, given the form of solution, there are at most \(l + 1\) adjustable parameters (where \(l\) is the number of training samples), but this seems to be begging the question\(^\text{13}\). It must be something to do with our requirement of maximum margin hyperplanes that is saving the day. As we shall see below, a strong case can be made for this claim.

Since the mapped surface is of intrinsic dimension \(\dim(\mathcal{L})\), unless \(\dim(\mathcal{L}) = \dim(\mathcal{H})\), it is obvious that the mapping cannot be onto (surjective). It also need not be one to one (bijective): consider \(x_1 \rightarrow -x_1\), \(x_2 \rightarrow -x_2\) in Eq. (62). The image of \(\Phi\) need not itself be a vector space: again, considering the above simple quadratic example, the vector \(-\Phi(x)\) is not in the image of \(\Phi\) unless \(x = 0\). Further, a little playing with the inhomogeneous kernel

\[
K(x_i, x_j) = (x_i \cdot x_j + 1)^2
\]

will convince you that the corresponding \(\Phi\) can map two vectors that are linearly dependent in \(\mathcal{L}\) onto two vectors that are linearly independent in \(\mathcal{H}\).

So far we have considered cases where \(\Phi\) is done implicitly. One can equally well turn things around and start with \(\Phi\), and then construct the corresponding kernel. For example (Vapnik, 1996), if \(\mathcal{L} = \mathbb{R}^1\), then a Fourier expansion in the data \(x\), cut off after \(N\) terms, has the form

\[
f(x) = \frac{a_0}{2} + \sum_{r=1}^{N} (a_{1r} \cos(rx) + a_{2r} \sin(rx))
\]

and this can be viewed as a dot product between two vectors in \(\mathbb{R}^{2N+1}\): \(\mathbf{a} = (\frac{a_0}{\sqrt{2}}, a_{11}, \ldots, a_{21}, \ldots)\), and the mapped \(\Phi(x) = (\frac{1}{\sqrt{2}}, \cos(x), \cos(2x), \ldots, \sin(x), \sin(2x), \ldots)\). Then the corresponding (Dirichlet) kernel can be computed in closed form:

\[
\Phi(x_i) \cdot \Phi(x_j) = K(x_i, x_j) = \frac{\sin((N + 1/2)(x_i - x_j))}{2\sin((x_i - x_j)/2)}.
\]

This is easily seen as follows: letting \(\delta \equiv x_i - x_j\),

\[
\Phi(x_i) \cdot \Phi(x_j) = \frac{1}{2} + \sum_{r=1}^{N} \cos(rx_i) \cos(rx_j) + \sin(rx_i) \sin(rx_j)
\]

\[
= \frac{1}{2} + \sum_{r=0}^{N} \cos(r\delta) = \frac{1}{2} + \text{Re} \left\{ \sum_{r=0}^{N} e^{ir\delta} \right\}
\]

\[
= \frac{1}{2} + \text{Re} \left\{ (1 - e^{i(N+1)\delta})/(1 - e^{i\delta}) \right\}
\]

\[
= (\sin((N + 1/2)\delta)) / 2\sin(\delta/2).
\]

Finally, it is clear that the above implicit mapping trick will work for any algorithm in which the data only appear as dot products (for example, the nearest neighbor algorithm). This fact has been used to derive a nonlinear version of principal component analysis by (Schölkopf, Smola and Müller, 1998b); it seems likely that this trick will continue to find uses elsewhere.
4.3. Some Examples of Nonlinear SVMs

The first kernels investigated for the pattern recognition problem were the following:

\[ K(x, y) = (x \cdot y + 1)^p \]  \hspace{1cm} (74)

\[ K(x, y) = e^{-||x-y||^2/2\sigma^2} \]  \hspace{1cm} (75)

\[ K(x, y) = \tanh(\kappa x \cdot y - \delta) \]  \hspace{1cm} (76)

Eq. (74) results in a classifier that is a polynomial of degree \( p \) in the data; Eq. (75) gives a Gaussian radial basis function classifier, and Eq. (76) gives a particular kind of two-layer sigmoidal neural network. For the RBF case, the number of centers (\( N_S \) in Eq. (61)), the centers themselves (the \( s_i \)), the weights (\( \alpha_i \)), and the threshold (\( b \)) are all produced automatically by the SVM training and give excellent results compared to classical RBFs, for the case of Gaussian RBFs (Schölkopf et al, 1997). For the neural network case, the first layer consists of \( N_S \) sets of weights, each set consisting of \( d_L \) (the dimension of the data) weights, and the second layer consists of \( N_S \) weights (the \( \alpha_i \)), so that an evaluation simply requires taking a weighted sum of sigmoids, themselves evaluated on dot products of the test data with the support vectors. Thus for the neural network case, the architecture (number of weights) is determined by SVM training.

Note, however, that the hyperbolic tangent kernel only satisfies Mercer’s condition for certain values of the parameters \( \kappa \) and \( \delta \) (and of the data \( ||x||^2 \)). This was first noticed experimentally (Vapnik, 1995); however some necessary conditions on these parameters for positivity are now known.\(^14\)

Figure 9 shows results for the same pattern recognition problem as that shown in Figure 7, but where the kernel was chosen to be a cubic polynomial. Notice that, even though the number of degrees of freedom is higher, for the linearly separable case (left panel), the solution is roughly linear, indicating that the capacity is being controlled; and that the linearly non-separable case (right panel) has become separable.

\[Figure 9.\] Degree 3 polynomial kernel. The background colour shows the shape of the decision surface.

Finally, note that although the SVM classifiers described above are binary classifiers, they are easily combined to handle the multiclass case. A simple, effective combination trains
4.4. Global Solutions and Uniqueness

When is the solution to the support vector training problem global, and when is it unique? By “global”, we mean that there exists no other point in the feasible region at which the objective function takes a lower value. We will address two kinds of ways in which uniqueness may not hold: solutions for which \( \{w, b\} \) are themselves unique, but for which the expansion of \( w \) in Eq. (46) is not; and solutions whose \( \{w, b\} \) differ. Both are of interest: even if the pair \( \{w, b\} \) is unique, if the \( \alpha_i \) are not, there may be equivalent expansions of \( w \) which require fewer support vectors (a trivial example of this is given below), and which therefore require fewer instructions during test phase.

It turns out that every local solution is also global. This is a property of any convex programming problem (Fletcher, 1987). Furthermore, the solution is guaranteed to be unique if the objective function (Eq. (43)) is strictly convex, which in our case means that the Hessian must be positive definite (note that for quadratic objective functions \( F \), the Hessian is positive definite if and only if \( F \) is strictly convex; this is not true for non-quadratic \( F \)): there, a positive definite Hessian implies a strictly convex objective function, but not vice versa (consider \( F = x^4 \) (Fletcher, 1987)). However, even if the Hessian is positive semidefinite, the solution can still be unique: consider two points along the real line with coordinates \( x_1 = 1 \) and \( x_2 = 2 \), and with polarities \(+\) and \(-\). Here the Hessian is positive semidefinite, but the solution \( (w = -2, \ b = 3, \ \xi_i = 0 \) in Eqs. (40), (41), (42)) is unique. It is also easy to find solutions which are not unique in the sense that the \( \alpha_i \) in the expansion of \( w \) are not unique: for example, consider the problem of four separable points on a square in \( \mathbb{R}^2 \): \( x_1 = [1, 1], x_2 = [-1, 1], x_3 = [-1, -1] \) and \( x_4 = [1, -1] \), with polarities \([+,-,-,+]\) respectively. One solution is \( w = [1, 0], b = 0, \ \alpha = [0.25, 0.25, 0.25, 0.25] \); another has the same \( w \) and \( b \), but \( \alpha = [0.5, 0.5, 0, 0] \) (note that both solutions satisfy the constraints \( \alpha_i > 0 \) and \( \sum_i \alpha_i y_i = 0 \)). When can this occur in general? Given some solution \( \alpha \), choose an \( \alpha' \) which is in the null space of the Hessian \( H_{ij} = y_i y_j x_i \cdot x_j \), and require that \( \alpha' \) be orthogonal to the vector all of whose components are 1. Then adding \( \alpha' \) to \( \alpha \) in Eq. (43) will leave \( L_D \) unchanged. If \( 0 \leq \alpha_i + \alpha_i' \leq C \) and \( \alpha' \) satisfies Eq. (45), then \( \alpha + \alpha' \) is also a solution\(^\text{15}\).

How about solutions where the \( \{w, b\} \) are themselves not unique? (We emphasize that this can only happen in principle if the Hessian is not positive definite, and even then, the solutions are necessarily global). The following very simple theorem shows that if non-unique solutions occur, then the solution at one optimal point is continuously deformable into the solution at the other optimal point, in such a way that all intermediate points are also solutions.

**Theorem 2** Let the variable \( X \) stand for the pair of variables \( \{w, b\} \). Let the Hessian for the problem be positive semidefinite, so that the objective function is convex. Let \( X_0 \) and \( X_1 \) be two points at which the objective function attains its minimal value. Then there exists a path \( X = X(\tau) = (1 - \tau)X_0 + \tau X_1, \ \tau \in [0, 1], \) such that \( X(\tau) \) is a solution for all \( \tau \).

**Proof:** Let the minimum value of the objective function be \( F_{\text{min}} \). Then by assumption, \( F(X_0) = F(X_1) = F_{\text{min}} \). By convexity of \( F \), \( F(X(\tau)) \leq (1 - \tau)F(X_0) + \tau F(X_1) = F_{\text{min}} \). Furthermore, by linearity, the \( X(\tau) \) satisfy the constraints Eq. (40), (41): explicitly (again combining both constraints into one):
\[
y_i(w_r \cdot x_i + b_r) = y_i((1 - \tau)(w_0 \cdot x_i + b_0) + \tau(w_1 \cdot x_i + b_1)) \\
\geq (1 - \tau)(1 - \xi_i) + \tau(1 - \xi_i) = 1 - \xi_i
\] (77)

Although simple, this theorem is quite instructive. For example, one might think that the problems depicted in Figure 10 have several different optimal solutions (for the case of linear support vector machines). However, since one cannot smoothly move the hyperplane from one proposed solution to another without generating hyperplanes which are not solutions, we know that these proposed solutions are in fact not solutions at all. In fact, for each of these cases, the optimal unique solution is at \( w = 0 \), with a suitable choice of \( b \) (which has the effect of assigning the same label to all the points). Note that this is a perfectly acceptable solution to the classification problem: any proposed hyperplane (with \( w \neq 0 \)) will cause the primal objective function to take a higher value.

![Figure 10: Two problems, with proposed (incorrect) non-unique solutions.](image)

Finally, note that the fact that SVM training always finds a global solution is in contrast to the case of neural networks, where many local minima usually exist.

5. **Methods of Solution**

The support vector optimization problem can be solved analytically only when the number of training data is very small, or for the separable case when it is known beforehand which of the training data become support vectors (as in Sections 3.3 and 6.2). Note that this can happen when the problem has some symmetry (Section 3.3), but that it can also happen when it does not (Section 6.2). For the general analytic case, the worst case computational complexity is of order \( N_S^2 \) (inversion of the Hessian), where \( N_S \) is the number of support vectors, although the two examples given both have complexity of \( O(1) \).

However, in most real world cases, Equations (43) (with dot products replaced by kernels), (44), and (45) must be solved numerically. For small problems, any general purpose optimization package that solves linearly constrained convex quadratic programs will do. A good survey of the available solvers, and where to get them, can be found\(^{16} \) in (Moré and Wright, 1983).

For larger problems, a range of existing techniques can be brought to bear. A full exploration of the relative merits of these methods would fill another tutorial. Here we just describe the general issues, and for concreteness, give a brief explanation of the technique we currently use. Below, a “face” means a set of points lying on the boundary of the feasible region, and an “active constraint” is a constraint for which the equality holds. For more
on nonlinear programming techniques see (Fletcher, 1987; Mangasarian, 1969; McCormick, 1983).

The basic recipe is to (1) note the optimality (KKT) conditions which the solution must satisfy, (2) define a strategy for approaching optimality by uniformly increasing the dual objective function subject to the constraints, and (3) decide on a decomposition algorithm so that only portions of the training data need be handled at a given time (Boser, Guyon and Vapnik, 1992; Osuna, Freund and Girosi, 1997a). We give a brief description of some of the issues involved. One can view the problem as requiring the solution of a sequence of equality constrained problems. A given equality constrained problem can be solved in one step by using the Newton method (although this requires storage for a factorization of the projected Hessian), or in at most $l$ steps using conjugate gradient ascent (Press et al., 1992) (where $l$ is the number of data points for the problem currently being solved: no extra storage is required). Some algorithms move within a given face until a new constraint is encountered, in which case the algorithm is restarted with the new constraint added to the list of equality constraints. This method has the disadvantage that only one new constraint is made active at a time. “Projection methods” have also been considered (Moré, 1991), where a point outside the feasible region is computed, and then line searches and projections are done so that the actual move remains inside the feasible region. This approach can add several new constraints at once. Note that in both approaches, several active constraints can become inactive in one step. In all algorithms, only the essential part of the Hessian (the columns corresponding to $\alpha_i \neq 0$) need be computed (although some algorithms do compute the whole Hessian). For the Newton approach, one can also take advantage of the fact that the Hessian is positive semidefinite by diagonalizing it with the Bunch-Kaufman algorithm (Bunch and Kaufman, 1977; Bunch and Kaufman, 1980) (if the Hessian were indefinite, it could still be easily reduced to 2x2 block diagonal form with this algorithm). In this algorithm, when a new constraint is made active or inactive, the factorization of the projected Hessian is easily updated (as opposed to recomputing the factorization from scratch). Finally, in interior point methods, the variables are essentially rescaled so as to always remain inside the feasible region. An example is the “LOQO” algorithm of (Vanderbei, 1994a; Vanderbei, 1994b), which is a primal-dual path following algorithm. This last method is likely to be useful for problems where the number of support vectors as a fraction of training sample size is expected to be large.

We briefly describe the core optimization method we currently use$^{17}$. It is an active set method combining gradient and conjugate gradient ascent. Whenever the objective function is computed, so is the gradient, at very little extra cost. In phase 1, the search directions $s$ are along the gradient. The nearest face along the search direction is found. If the dot product of the gradient there with $s$ indicates that the maximum along $s$ lies between the current point and the nearest face, the optimal point along the search direction is computed analytically (note that this does not require a line search), and phase 2 is entered. Otherwise, we jump to the new face and repeat phase 1. In phase 2, Polak-Ribiere conjugate gradient ascent (Press et al., 1992) is done, until a new face is encountered (in which case phase 1 is re-entered) or the stopping criterion is met. Note the following:

- Search directions are always projected so that the $\alpha_i$ continue to satisfy the equality constraint Eq. (45). Note that the conjugate gradient algorithm will still work; we are simply searching in a subspace. However, it is important that this projection is implemented in such a way that not only is Eq. (45) met (easy), but also so that the angle between the resulting search direction, and the search direction prior to projection, is minimized (not quite so easy).
• We also use a “sticky faces” algorithm: whenever a given face is hit more than once, the search directions are adjusted so that all subsequent searches are done within that face. All “sticky faces” are reset (made “non-sticky”) when the rate of increase of the objective function falls below a threshold.

• The algorithm stops when the fractional rate of increase of the objective function $F$ falls below a tolerance (typically 1e-10, for double precision). Note that one can also use as stopping criterion the condition that the size of the projected search direction falls below a threshold. However, this criterion does not handle scaling well.

• In my opinion the hardest thing to get right is handling precision problems correctly everywhere. If this is not done, the algorithm may not converge, or may be much slower than it needs to be.

A good way to check that your algorithm is working is to check that the solution satisfies all the Karush-Kuhn-Tucker conditions for the primal problem, since these are necessary and sufficient conditions that the solution be optimal. The KKT conditions are Eqs. (48) through (56), with dot products between data vectors replaced by kernels wherever they appear (note $\mathbf{w}$ must be expanded as in Eq. (48) first, since $\mathbf{w}$ is not in general the mapping of a point in $\mathcal{L}$). Thus to check the KKT conditions, it is sufficient to check that the $\alpha_i$ satisfy $0 \leq \alpha_i \leq C$, that the equality constraint (49) holds, that all points for which $0 \leq \alpha_i < C$ satisfy Eq. (51) with $\xi_i = 0$, and that all points with $\alpha_i = C$ satisfy Eq. (51) for some $\xi_i \geq 0$. These are sufficient conditions for all the KKT conditions to hold: note that by doing this we never have to explicitly compute the $\xi_i$ or $\mu_i$, although doing so is trivial.

5.1. Complexity, Scalability, and Parallelizability

Support vector machines have the following very striking property. Both training and test functions depend on the data only through the kernel functions $K(x_i, x_j)$. Even though it corresponds to a dot product in a space of dimension $d_H$, where $d_H$ can be very large or infinite, the complexity of computing $K$ can be far smaller. For example, for kernels of the form $K = (x_i, x_j)^p$, a dot product in $\mathcal{H}$ would require of order $\binom{d_H}{p}$ operations, whereas the computation of $K(x_i, x_j)$ requires only $O(d_L)$ operations (recall $d_L$ is the dimension of the data). It is this fact that allows us to construct hyperplanes in these very high dimensional spaces yet still be left with a tractable computation. Thus SVMs circumvent both forms of the “curse of dimensionality”: the proliferation of parameters causing intractable complexity, and the proliferation of parameters causing overfitting.

5.1.1. Training For concreteness, we will give results for the computational complexity of one the above training algorithms (Bunch-Kaufman)\textsuperscript{18} (Kaufman, 1998). These results assume that different strategies are used in different situations. We consider the problem of training on just one “chunk” (see below). Again let $l$ be the number of training points, $N_S$ the number of support vectors (SVs), and $d_L$ the dimension of the input data. In the case where most SVs are not at the upper bound, and $N_S/l << 1$, the number of operations $\mathcal{C}$ is $O(N_S^3 + (N_S^2)l + N_Sd_Ll)$. If instead $N_S/l \approx 1$, then $\mathcal{C}$ is $O(N_S^3 + N_Sl + N_Sd_Ll)$ (basically by starting in the interior of the feasible region). For the case where most SVs are at the upper bound, and $N_S/l << 1$, then $\mathcal{C}$ is $O(N_S^2 + N_Sd_Ll)$. Finally, if most SVs are at the upper bound, and $N_S/l \approx 1$, we have $\mathcal{C}$ of $O(D_L l^3)$.

For larger problems, two decomposition algorithms have been proposed to date. In the “chunking” method (Boser, Guyon and Vapnik, 1992), one starts with a small, arbitrary
subset of the data and trains on that. The rest of the training data is tested on the resulting classifier, and a list of the errors is constructed, sorted by how far on the wrong side of the margin they lie (i.e. how egregiously the KKT conditions are violated). The next chunk is constructed from the first \( N \) of these, combined with the \( N_S \) support vectors already found, where \( N + N_S \) is decided heuristically (a chunk size that is allowed to grow too quickly or too slowly will result in slow overall convergence). Note that vectors can be dropped from a chunk, and that support vectors in one chunk may not appear in the final solution. This process is continued until all data points are found to satisfy the KKT conditions.

The above method requires that the number of support vectors \( N_S \) be small enough so that a Hessian of size \( N_S \) by \( N_S \) will fit in memory. An alternative decomposition algorithm has been proposed which overcomes this limitation (Osuna, Freund and Girosi, 1997b). Again, in this algorithm, only a small portion of the training data is trained on at a given time, and furthermore, only a subset of the support vectors need be in the "working set" (i.e. that set of points whose \( \alpha \)'s are allowed to vary). This method has been shown to be able to easily handle a problem with 110,000 training points and 100,000 support vectors. However, it must be noted that the speed of this approach relies on many of the support vectors having corresponding Lagrange multipliers \( \alpha_i \) at the upper bound, \( \alpha_i = C \).

These training algorithms may take advantage of parallel processing in several ways. First, all elements of the Hessian itself can be computed simultaneously. Second, each element often requires the computation of dot products of training data, which could also be parallelized. Third, the computation of the objective function, or gradient, which is a speed bottleneck, can be parallelized (it requires a matrix multiplication). Finally, one can envision parallelizing at a higher level, for example by training on different chunks simultaneously. Schemes such as these, combined with the decomposition algorithm of (Osuna, Freund and Girosi, 1997b), will be needed to make very large problems (i.e. \( \gg 100,000 \) support vectors, with many not at bound), tractable.

5.1.2. Testing In test phase, one must simply evaluate Eq. (61), which will require \( O(MN_S) \) operations, where \( M \) is the number of operations required to evaluate the kernel. For dot product and RBF kernels, \( M \) is \( O(d_L) \), the dimension of the data vectors. Again, both the evaluation of the kernel and of the sum are highly parallelizable procedures.

In the absence of parallel hardware, one can still speed up test phase by a large factor, as described in Section 9.

6. The VC Dimension of Support Vector Machines

We now show that the VC dimension of SVMs can be very large (even infinite). We will then explore several arguments as to why, in spite of this, SVMs usually exhibit good generalization performance. However it should be emphasized that these are essentially plausibility arguments. Currently there exists no theory which guarantees that a given family of SVMs will have high accuracy on a given problem.

We will call any kernel that satisfies Mercer's condition a positive kernel, and the corresponding space \( \mathcal{H} \) the embedding space. We will also call any embedding space with minimal dimension for a given kernel a "minimal embedding space". We have the following

**Theorem 3** Let \( K \) be a positive kernel which corresponds to a minimal embedding space \( \mathcal{H} \). Then the VC dimension of the corresponding support vector machine (where the error penalty \( C \) in Eq. (44) is allowed to take all values) is \( \dim(\mathcal{H}) + 1 \).
Proof: If the minimal embedding space has dimension $d_H$, then $d_H$ points in the image of $\mathcal{L}$ under the mapping $\Phi$ can be found whose position vectors in $\mathcal{H}$ are linearly independent. From Theorem 1, these vectors can be shattered by hyperplanes in $\mathcal{H}$. Thus by either restricting ourselves to SVMs for the separable case (Section 3.1), or for which the error penalty $C$ is allowed to take all values (so that, if the points are linearly separable, a $C$ can be found such that the solution does indeed separate them), the family of support vector machines with kernel $K$ can also shatter these points, and hence has VC dimension $d_H + 1$.

Let's look at two examples.

6.1. The VC Dimension for Polynomial Kernels

Consider an SVM with homogeneous polynomial kernel, acting on data in $\mathbb{R}^{d_L}$:

$$K(x_1, x_2) = (x_1 \cdot x_2)^p, \quad x_1, x_2 \in \mathbb{R}^{d_L}$$

(78)

As in the case when $d_L = 2$ and the kernel is quadratic (Section 4), one can explicitly construct the map $\Phi$. Letting $z_i = x_1^i x_2^i$, so that $K(x_1, x_2) = (z_1 + \cdots + z_{d_L})^p$, we see that each dimension of $\mathcal{H}$ corresponds to a term with given powers of the $z_i$ in the expansion of $K$. In fact if we choose to label the components of $\Phi(x)$ in this manner, we can explicitly write the mapping for any $p$ and $d_L$:

$$\Phi_{r_1 \cdots r_{d_L}}(x) = \sqrt{\frac{p!}{r_1! \cdots r_{d_L}!}} z_1^{r_1} z_2^{r_2} \cdots z_{d_L}^{r_{d_L}}, \quad \sum_{i=1}^{d_L} r_i = p, \quad r_i \geq 0$$

(79)

This leads to

Theorem 4 If the space in which the data live has dimension $d_L$ (i.e. $\mathcal{L} = \mathbb{R}^{d_L}$), the dimension of the minimal embedding space, for homogeneous polynomial kernels of degree $p$ ($K(x_1, x_2) = (x_1 \cdot x_2)^p, \ x_1, \ x_2 \in \mathbb{R}^{d_L}$), is $\binom{d_L + p - 1}{p}$.

(The proof is in the Appendix). Thus the VC dimension of SVMs with these kernels is $\binom{d_L + p - 1}{p} + 1$. As noted above, this gets very large very quickly.

6.2. The VC Dimension for Radial Basis Function Kernels

Theorem 5 Consider the class of Mercer kernels for which $K(x_1, x_2) \to 0$ as $\|x_1 - x_2\| \to \infty$, and for which $K(x, x)$ is $O(1)$, and assume that the data can be chosen arbitrarily from $\mathbb{R}^d$. Then the family of classifiers consisting of support vector machines using these kernels, and for which the error penalty is allowed to take all values, has infinite VC dimension.

Proof: The kernel matrix, $K_{ij} = K(x_i, x_j)$, is a Gram matrix (a matrix of dot products: see (Horn, 1985)) in $\mathcal{H}$. Clearly we can choose training data such that all off-diagonal elements $K_{ij}$ can be made arbitrarily small, and by assumption all diagonal elements $K_{ii}$ are of $O(1)$. The matrix $K$ is then of full rank; hence the set of vectors, whose dot products in $\mathcal{H}$ form $K$, are linearly independent (Horn, 1985); hence, by Theorem 1, the points can be shattered by hyperplanes in $\mathcal{H}$, and hence also by support vector machines with sufficiently large error penalty. Since this is true for any finite number of points, the VC dimension of these classifiers is infinite. 

■
Note that the assumptions in the theorem are stronger than necessary (they were chosen to make the connection to radial basis functions clear). In fact it is only necessary that \( l \) training points can be chosen such that the rank of the matrix \( K_{ij} \) increases without limit as \( l \) increases. For example, for Gaussian RBF kernels, this can also be accomplished (even for training data restricted to lie in a bounded subset of \( \mathbb{R}^{d_L} \)) by choosing small enough RBF widths. However in general the VC dimension of SVM RBF classifiers can certainly be finite, and indeed, for data restricted to lie in a bounded subset of \( \mathbb{R}^{d_L} \), choosing restrictions on the RBF widths is a good way to control the VC dimension.

This case gives us a second opportunity to present a situation where the SVM solution can be computed analytically, which also amounts to a second, constructive proof of the Theorem. For concreteness we will take the case for Gaussian RBF kernels of the form
\[
K(x_1, x_2) = e^{-||x_1 - x_2||^2 / 2\sigma^2}.
\]
Let us choose training points such that the smallest distance between any pair of points is much larger than the width \( \sigma \). Consider the decision function evaluated on the support vector \( s_j \):
\[
f(s_j) = \sum_i \alpha_i y_i e^{-||s_i - s_j||^2 / 2\sigma^2} + b.
\]

The sum on the right hand side will then be largely dominated by the term \( i = j \); in fact the ratio of that term to the contribution from the rest of the sum can be made arbitrarily large by choosing the training points to be arbitrarily far apart. In order to find the SVM solution, we again assume for the moment that every training point becomes a support vector, and we work with SVMs for the separable case (Section 3.1) (the same argument will hold for SVMs for the non-separable case if \( C \) in Eq. (44) is allowed to take large enough values). Since all points are support vectors, the equalities in Eqs. (10), (11) will hold for them. Let there be \( N_+ \) (\( N_- \)) positive (negative) polarity points. We further assume that all positive (negative) polarity points have the same value \( \alpha_+ \) (\( \alpha_- \)) for their Lagrange multiplier. (We will know that this assumption is correct if it delivers a solution which satisfies all the KKT conditions and constraints). Then Eqs. (19), applied to all the training data, and the equality constraint Eq. (18), become
\[
\begin{align*}
\alpha_+ + b &= 1 \\
-\alpha_- + b &= -1 \\
N_+ \alpha_+ - N_- \alpha_- &= 0
\end{align*}
\]
which are satisfied by
\[
\begin{align*}
\alpha_+ &= \frac{2N_-}{N_- + N_+} \\
\alpha_- &= \frac{2N_+}{N_- + N_+} \\
b &= \frac{N_+ - N_-}{N_- + N_+}
\end{align*}
\]
Thus, since the resulting \( \alpha_i \) are also positive, all the KKT conditions and constraints are satisfied, and we must have found the global solution (with zero training errors). Since the number of training points, and their labeling, is arbitrary, and they are separated without error, the VC dimension is infinite.

The situation is summarized schematically in Figure 11.
Now we are left with a striking conundrum. Even though their VC dimension is infinite (if the data is allowed to take all values in $\mathbb{R}^d$), SVM RBFs can have excellent performance (Schölkopf et al, 1997). A similar story holds for polynomial SVMs. How come?

7. The Generalization Performance of SVMs

In this Section we collect various arguments and bounds relating to the generalization performance of SVMs. We start by presenting a family of SVM-like classifiers for which structural risk minimization can be rigorously implemented, and which will give us some insight as to why maximizing the margin is so important.

7.1. VC Dimension of Gap Tolerant Classifiers

Consider a family of classifiers (i.e. a set of functions $\Phi$ on $\mathbb{R}^d$) which we will call “gap tolerant classifiers.” A particular classifier $\phi \in \Phi$ is specified by the location and diameter of a ball in $\mathbb{R}^d$, and by two hyperplanes, with parallel normals, also in $\mathbb{R}^d$. Call the set of points lying between, but not on, the hyperplanes the “margin set.” The decision functions $\phi$ are defined as follows: points that lie inside the ball, but not in the margin set, are assigned class $\{\pm 1\}$, depending on which side of the margin set they fall. All other points are simply defined to be “correct”, that is, they are not assigned a class by the classifier, and do not contribute to any risk. The situation is summarized, for $d = 2$, in Figure 12. This rather odd family of classifiers, together with a condition we will impose on how they are trained, will result in systems very similar to SVMs, and for which structural risk minimization can be demonstrated. A rigorous discussion is given in the Appendix.

Label the diameter of the ball $D$ and the perpendicular distance between the two hyperplanes $M$. The VC dimension is defined as before to be the maximum number of points that can be shattered by the family, but by “shattered” we mean that the points can occur as errors in all possible ways (see the Appendix for further discussion). Clearly we can control the VC dimension of a family of these classifiers by controlling the minimum margin $M$ and maximum diameter $D$ that members of the family are allowed to assume. For example, consider the family of gap tolerant classifiers in $\mathbb{R}^2$ with diameter $D = 2$, shown in Figure 12. Those with margin satisfying $M \leq 3/2$ can shatter three points; if $3/2 < M < 2$, they can shatter two; and if $M \geq 2$, they can shatter only one. Each of these three families of
classifiers corresponds to one of the sets of classifiers in Figure 4, with just three nested subsets of functions, and with \( h_1 = 1 \), \( h_2 = 2 \), and \( h_3 = 3 \).

![Figure 12](image.png)

*Figure 12. A gap tolerant classifier on data in \( \mathbb{R}^2 \).*

These ideas can be used to show how gap tolerant classifiers implement structural risk minimization. The extension of the above example to spaces of arbitrary dimension is encapsulated in a (modified) theorem of (Vapnik, 1995):

**Theorem 6** For data in \( \mathbb{R}^d \), the VC dimension \( h \) of gap tolerant classifiers of minimum margin \( M_{\min} \) and maximum diameter \( D_{\max} \) is bounded above by \( \min\{ [D_{\max}^2/M_{\min}^2], d \} + 1 \).

For the proof we assume the following lemma, which in (Vapnik, 1979) is held to follow from symmetry arguments:

**Lemma:** Consider \( n \leq d + 1 \) points lying in a ball \( B \in \mathbb{R}^d \). Let the points be shatterable by gap tolerant classifiers with margin \( M \). Then in order for \( M \) to be maximized, the points must lie on the vertices of an \((n-1)\)-dimensional symmetric simplex, and must also lie on the surface of the ball.

**Proof:** We need only consider the case where the number of points \( n \) satisfies \( n \leq d + 1 \). \( n > d + 1 \) points will not be shatterable, since the VC dimension of oriented hyperplanes in \( \mathbb{R}^d \) is \( d + 1 \), and any distribution of points which can be shattered by a gap tolerant classifier can also be shattered by an oriented hyperplane; this also shows that \( h \leq d + 1 \). Again we consider points on a sphere of diameter \( D \), where the sphere itself is of dimension \( d - 2 \). We will need two results from Section 3.3, namely (1) if \( n \) is even, we can find a distribution of \( n \) points (the vertices of the \((n-1)\)-dimensional symmetric simplex) which can be shattered by gap tolerant classifiers if \( D_{\max}^2/M_{\min}^2 = n - 1 \), and (2) if \( n \) is odd, we can find a distribution of \( n \) points which can be so shattered if \( D_{\max}^2/M_{\min}^2 = (n - 1)^2(n + 1)/n^2 \).

If \( n \) is even, at most \( n \) points can be shattered whenever

\[
n - 1 \leq D_{\max}^2/M_{\min}^2 < n. \tag{83}
\]
Thus for \( n \) even the maximum number of points that can be shattered may be written 
\[ D_{\text{max}}^2/M_{\text{min}}^2 + 1. \]

If \( n \) is odd, at most \( n \) points can be shattered when 
\[ D_{\text{max}}^2/M_{\text{min}}^2 = (n - 1)^2(n + 1)/n^2. \] 
However, the quantity on the right hand side satisfies 
\[ n - 2 < (n - 1)^2(n + 1)/n^2 < n - 1 \] 
for all integer \( n > 1 \). Thus for \( n \) odd the largest number of points that can be shattered is 
certainly bounded above by \( D_{\text{max}}^2/M_{\text{min}}^2 + 1 \), and from the above this bound is also
satisfied when \( n \) is even. Hence in general the VC dimension \( h \) of gap tolerant classifiers
must satisfy
\[ h \leq \left\lfloor \frac{D_{\text{max}}^2}{M_{\text{min}}^2} \right\rfloor + 1. \] 

This result, together with \( h \leq d + 1 \), concludes the proof. \( \blacksquare \)

7.2. Gap Tolerant Classifiers, Structural Risk Minimization, and SVMs

Let’s see how we can do structural risk minimization with gap tolerant classifiers. We need
only consider that subset of the \( \Phi \), call it \( \Phi_S \), for which training “succeeds”, where by success
we mean that all training data are assigned a label \( \in \{ \pm 1 \} \) (note that these labels do not
have to coincide with the actual labels, i.e. training errors are allowed). Within \( \Phi_S \), find
the subset which gives the fewest training errors - call this number of errors \( N_{\text{min}} \). Within
that subset, find the function \( \phi \) which gives maximum margin (and hence the lowest bound
on the VC dimension). Note the value of the resulting risk bound (the right hand side of
Eq. (3), using the bound on the VC dimension in place of the VC dimension). Next, within
\( \Phi_S \), find that subset which gives \( N_{\text{min}} + 1 \) training errors. Again, within that subset, find
the \( \phi \) which gives the maximum margin, and note the corresponding risk bound. Iterate,
and take that classifier which gives the overall minimum risk bound.

An alternative approach is to divide the functions \( \Phi \) into nested subsets \( \Phi_i, i \in \mathcal{Z}, i \geq 1 \),
as follows: all \( \phi \in \Phi_i \) have \( \{ D, M \} \) satisfying \( [D^2/M^2] \leq i \). Thus the family of functions
in \( \Phi_i \) has VC dimension bounded above by \( \min(i, d) + 1 \). Note also that \( \Phi_i \subset \Phi_{i+1} \). SRM
then proceeds by taking that \( \phi \) for which training succeeds in each subset and for which
the empirical risk is minimized in that subset, and again, choosing that \( \phi \) which gives the
lowest overall risk bound.

Note that it is essential to these arguments that the bound (3) holds for any chosen decision
function, not just the one that minimizes the empirical risk (otherwise eliminating solutions
for which some training point \( x \) satisfies \( \phi(x) = 0 \) would invalidate the argument).

The resulting gap tolerant classifier is in fact a special kind of support vector machine
which simply does not count data falling outside the sphere containing all the training data,
or inside the separating margin, as an error. It seems very reasonable to conclude that
support vector machines, which are trained with very similar objectives, also gain a similar
kind of capacity control from their training. However, a gap tolerant classifier is not an
SVM, and so the argument does not constitute a rigorous demonstration of structural risk
minimization for SVMs. The original argument for structural risk minimization for SVMs is
known to be flawed, since the structure there is determined by the data (see (Vapnik, 1995),
Section 5.11). I believe that there is a further subtle problem with the original argument.
The structure is defined so that no training points are members of the margin set. However,
one must still specify how test points that fall into the margin are to be labeled. If one simply
assigns the same, fixed class to them (say +1), then the VC dimension will be higher\(^{21}\) than the bound derived in Theorem 6. However, the same is true if one labels them all as errors (see the Appendix). If one labels them all as “correct”, one arrives at gap tolerant classifiers.

On the other hand, it is known how to do structural risk minimization for systems where the structure does depend on the data (Shawe-Taylor et al., 1996a; Shawe-Taylor et al., 1996b). Unfortunately the resulting bounds are much looser than the VC bounds above, which are already very loose (we will examine a typical case below where the VC bound is a factor of 100 higher than the measured test error). Thus at the moment structural risk minimization alone does not provide a rigorous explanation as to why SVMs often have good generalization performance. However, the above arguments strongly suggest that algorithms that minimize \(D^2/M^2\) can be expected to give better generalization performance. Further evidence for this is found in the following theorem of (Vapnik, 1998), which we quote without proof\(^{22}\):

**Theorem 7** For optimal hyperplanes passing through the origin, we have

\[
E[P(\text{error})] \leq \frac{E[D^2/M^2]}{I}
\]  

(86)

where \(P(\text{error})\) is the probability of error on the test set, the expectation on the left is over all training sets of size \(I - 1\), and the expectation on the right is over all training sets of size \(I\).

However, in order for these observations to be useful for real problems, we need a way to compute the diameter of the minimal enclosing sphere described above, for any number of training points and for any kernel mapping.

### 7.3. How to Compute the Minimal Enclosing Sphere

Again let \(\Phi\) be the mapping to the embedding space \(\mathcal{H}\). We wish to compute the radius of the smallest sphere in \(\mathcal{H}\) which encloses the mapped training data: that is, we wish to minimize \(R^2\) subject to

\[
\|\Phi(x_i) - C\|^2 \leq R^2 \quad \forall i
\]  

(87)

where \(C \in \mathcal{H}\) is the (unknown) center of the sphere. Thus introducing positive Lagrange multipliers \(\lambda_i\), the primal Lagrangian is

\[
L_P = R^2 - \sum_i \lambda_i (R^2 - \|\Phi(x_i) - C\|^2).
\]  

(88)

This is again a convex quadratic programming problem, so we can instead maximize the Wolfe dual

\[
L_D = \sum_i \lambda_i K(x_i, x_i) - \sum_{i,j} \lambda_i \lambda_j K(x_i, x_j)
\]  

(89)

(where we have again replaced \(\Phi(x_i) \cdot \Phi(x_j)\) by \(K(x_i, x_j)\)) subject to:

\[
\sum_i \lambda_i = 1 \quad (90)
\]

\[
\lambda_i \geq 0 \quad (91)
\]
with solution given by

$$C = \sum_i \lambda_i \Phi(x_i).$$  \hspace{1cm} (92)

Thus the problem is very similar to that of support vector training, and in fact the code for the latter is easily modified to solve the above problem. Note that we were in a sense “lucky”, because the above analysis shows us that there exists an expansion (92) for the center; there is no \textit{a priori} reason why we should expect that the center of the sphere in $\mathcal{H}$ should be expressible in terms of the mapped training data in this way. The same can be said of the solution for the support vector problem, Eq. (46). (Had we chosen some other geometrical construction, we might not have been so fortunate. Consider the smallest area equilateral triangle containing two given points in $\mathbb{R}^2$. If the points’ position vectors are linearly dependent, the center of the triangle cannot be expressed in terms of them.)

7.4. \textit{A Bound from Leave-One-Out}

(Vapnik, 1995) gives an alternative bound on the actual risk of support vector machines:

$$E[P(\text{error})] \leq \frac{E[\text{Number of support vectors}]}{\text{Number of training samples}},$$  \hspace{1cm} (93)

where $P(\text{error})$ is the actual risk for a machine trained on $l - 1$ examples, $E[P(\text{error})]$ is the expectation of the actual risk over all choices of training set of size $l - 1$, and $E[\text{Number of support vectors}]$ is the expectation of the number of support vectors over all choices of training sets of size $l$. It’s easy to see how this bound arises: consider the typical situation after training on a given training set, shown in Figure 13.

\begin{figure}[h]
\centering
\includegraphics[width=\linewidth]{figure13.png}
\caption{Support vectors (circles) can become errors (cross) after removal and re-training (the dotted line denotes the new decision surface).}
\end{figure}

We can get an estimate of the test error by removing one of the training points, re-training, and then testing on the removed point; and then repeating this, for all training points. From the support vector solution we know that removing any training points that are not support vectors (the latter include the errors) will have no effect on the hyperplane found. Thus the worst that can happen is that every support vector will become an error. Taking the expectation over all such training sets therefore gives an upper bound on the actual risk, for training sets of size $l - 1$. 
Although elegant, I have yet to find a use for this bound. There seem to be many situations where the actual error increases even though the number of support vectors decreases, so the intuitive conclusion (systems that give fewer support vectors give better performance) often seems to fail. Furthermore, although the bound can be tighter than that found using the estimate of the VC dimension combined with Eq. (3), it can at the same time be less predictive, as we shall see in the next Section.

7.5. VC, SV Bounds and the Actual Risk

Let us put these observations to some use. As mentioned above, training an SVM RBF classifier will automatically give values for the RBF weights, number of centers, center positions, and threshold. For Gaussian RBFs, there is only one parameter left: the RBF width (σ in Eq. (80)) (we assume here only one RBF width for the problem). Can we find the optimal value for that too, by choosing that σ which minimizes $D^2/M^2$? Figure 14 shows a series of experiments done on 28x28 NIST digit data, with 10,000 training points and 60,000 test points. The top curve in the left hand panel shows the VC bound (i.e. the bound resulting from approximating the VC dimension in Eq. (3) by Eq. (85)), the middle curve shows the bound from leave-one-out (Eq. (93)), and the bottom curve shows the measured test error. Clearly, in this case, the bounds are very loose. The right hand panel shows just the VC bound (the top curve, for $\sigma^2 > 200$), together with the test error, with the latter scaled up by a factor of 100 (note that the two curves cross). It is striking that the two curves have minima in the same place: thus in this case, the VC bound, although loose, seems to be nevertheless predictive. Experiments on digits 2 through 9 showed that the VC bound gave a minimum for which $\sigma^2$ was within a factor of two of that which minimized the test error (digit 1 was inconclusive). Interestingly, in those cases the VC bound consistently gave a lower prediction for $\sigma^2$ than that which minimized the test error. On the other hand, the leave-one-out bound, although tighter, does not seem to be predictive, since it had no minimum for the values of $\sigma^2$ tested.
8. Limitations

Perhaps the biggest limitation of the support vector approach lies in choice of the kernel. Once the kernel is fixed, SVM classifiers have only one user-chosen parameter (the error penalty), but the kernel is a very big rug under which to sweep parameters. Some work has been done on limiting kernels using prior knowledge (Schölkopf et al., 1998a; Burges, 1998), but the best choice of kernel for a given problem is still a research issue.

A second limitation is speed and size, both in training and testing. While the speed problem in test phase is largely solved in (Burges, 1996), this still requires two training passes. Training for very large datasets (millions of support vectors) is an unsolved problem.

Discrete data presents another problem, although with suitable rescaling excellent results have nevertheless been obtained (Joachims, 1997). Finally, although some work has been done on training a multiclass SVM in one step24, the optimal design for multiclass SVM classifiers is a further area for research.

9. Extensions

We very briefly describe two of the simplest, and most effective, methods for improving the performance of SVMs.

The virtual support vector method (Schölkopf, Burges and Vapnik, 1996; Burges and Schölkopf, 1997), attempts to incorporate known invariances of the problem (for example, translation invariance for the image recognition problem) by first training a system, and then creating new data by distorting the resulting support vectors (translating them, in the case mentioned), and finally training a new system on the distorted (and the undistorted) data. The idea is easy to implement and seems to work better than other methods for incorporating invariances proposed so far.

The reduced set method (Burges, 1996; Burges and Schölkopf, 1997) was introduced to address the speed of support vector machines in test phase, and also starts with a trained SVM. The idea is to replace the sum in Eq. (46) by a similar sum, where instead of support vectors, computed vectors (which are not elements of the training set) are used, and instead of the $\alpha_i$, a different set of weights are computed. The number of parameters is chosen beforehand to give the speedup desired. The resulting vector is still a vector in $\mathcal{H}$, and the parameters are found by minimizing the Euclidean norm of the difference between the original vector $w$ and the approximation to it. The same technique could be used for SVM regression to find much more efficient function representations (which could be used, for example, in data compression).

Combining these two methods gave a factor of 50 speedup (while the error rate increased from 1.0% to 1.1%) on the NIST digits (Burges and Schölkopf, 1997).

10. Conclusions

SVMs provide a new approach to the problem of pattern recognition (together with regression estimation and linear operator inversion) with clear connections to the underlying statistical learning theory. They differ radically from comparable approaches such as neural networks: SVM training always finds a global minimum, and their simple geometric interpretation provides fertile ground for further investigation. An SVM is largely characterized by the choice of its kernel, and SVMs thus link the problems they are designed for with a large body of existing work on kernel based methods. I hope that this tutorial will encourage some to explore SVMs for themselves.
Acknowledgments

I’m very grateful to P. Knirsch, C. Nohl, E. Osuna, E. Rietman, B. Schölkopf, Y. Singer, A. Smola, C. Stenard, and V. Vapnik, for their comments on the manuscript. Thanks also to the reviewers, and to the Editor, U. Fayad, for extensive, useful comments. Special thanks are due to V. Vapnik, under whose patient guidance I learned the ropes; to A. Smola and B. Schölkopf, for many interesting and fruitful discussions; and to J. Shawe-Taylor and D. Schuurmans, for valuable discussions on structural risk minimization.

Appendix

A.1. Proofs of Theorems

We collect here the theorems stated in the text, together with their proofs. The Lemma has a shorter proof using a “Theorem of the Alternative,” (Mangasarian, 1969) but we wished to keep the proofs as self-contained as possible.

Lemma 1 Two sets of points in $\mathbb{R}^n$ may be separated by a hyperplane if and only if the intersection of their convex hulls is empty.

Proof: We allow the notions of points in $\mathbb{R}^n$, and position vectors of those points, to be used interchangeably in this proof. Let $C_A, C_B$ be the convex hulls of two sets of points $A, B$ in $\mathbb{R}^n$. Let $A - B$ denote the set of points whose position vectors are given by $a - b, a \in A, b \in B$ (note that $A - B$ does not contain the origin), and let $C_A - C_B$ have the corresponding meaning for the convex hulls. Then showing that $A$ and $B$ are linearly separable (separable by a hyperplane) is equivalent to showing that the set $A - B$ is linearly separable from the origin $O$. For suppose the latter: then $\exists w \in \mathbb{R}^n, b \in \mathbb{R}, b < 0$ such that $x \cdot w + b > 0$ $\forall x \in A - B$. Now pick some $y \in B$, and denote the set of all points $a - b + y, a \in A, b \in B$ by $A - B + y$. Then $x \cdot w + b > y \cdot w$ $\forall x \in A - B + y$, and clearly $y \cdot w + b < y \cdot w$, so the sets $A - B + y$ and $y$ are linearly separable. Repeating this process shows that $A - B$ is linearly separable from the origin if and only if $A$ and $B$ are linearly separable.

We now show that, if $C_A \cap C_B = \emptyset$, then $C_A - C_B$ is linearly separable from the origin. Clearly $C_A - C_B$ does not contain the origin. Furthermore $C_A - C_B$ is convex, since $\forall x_1 = a_1 - b_1, x_2 = a_2 - b_2, \lambda \in [0, 1], a_1, a_2 \in C_A, b_1, b_2 \in C_B$, we have $(1 - \lambda)x_1 + \lambda x_2 = ((1 - \lambda)a_1 + \lambda a_2) - ((1 - \lambda)b_1 + \lambda b_2) \in C_A - C_B$. Hence it is sufficient to show that any convex set $S$, which does not contain $O$, is linearly separable from $O$. Let $x_{min} \in S$ be that point whose Euclidean distance from $O, ||x_{min}||$, is minimal. (Note there can be only one such point, since if there were two, the chord joining them, which also lies in $S$, would contain points closer to $O$.) We will show that $\forall x \in S, x \cdot x_{min} > 0$. Suppose $\exists x \in S$ such that $x \cdot x_{min} \leq 0$. Let $L$ be the line segment joining $x_{min}$ and $x$. Then convexity implies that $L \subset S$. Thus $O \notin L_s$ since by assumption $O \notin S$. Hence the three points $O, x$ and $x_{min}$ form an obtuse (or right) triangle, with obtuse (or right) angle occurring at the point $O$. Define $\hat{n} \equiv (x - x_{min})/||x - x_{min}||$. Then the distance from the closest point in $L$ to $O$ is $||x_{min}||^2 - (x_{min} \cdot \hat{n})^2$, which is less than $||x_{min}||^2$. Hence $x \cdot x_{min} > 0$ and $S$ is linearly separable from $O$. Thus $C_A - C_B$ is linearly separable from $O$, and a fortiori $A - B$ is linearly separable from $O$, and thus $A$ is linearly separable from $B$.

It remains to show that, if the two sets of points $A, B$ are linearly separable, the intersection of their convex hulls if empty. By assumption there exists a pair $w \in \mathbb{R}^n, b \in R$, such that $\forall a_i \in A, w \cdot a_i + b > 0$ and $\forall b_i \in B, w \cdot b_i + b < 0$. Consider a general point $x \in C_A$. It
may be written \( x = \sum \lambda_i a_i \), \( \sum \lambda_i = 1 \), \( 0 \leq \lambda_i \leq 1 \). Then \( w \cdot x + b = \sum \lambda_i (w \cdot a_i + b) > 0 \). Similarly, for points \( y \in C_B \), \( w \cdot y + b < 0 \). Hence \( C_A \cap C_B = \emptyset \), since otherwise we would be able to find a point \( x = y \) which simultaneously satisfies both inequalities.

\[ \textbf{Theorem 1:} \] Consider some set of \( m \) points in \( \mathbb{R}^n \). Choose any one of the points as origin. Then the \( m \) points can be shattered by oriented hyperplanes if and only if the position vectors of the remaining points are linearly independent.

\[ \textbf{Proof:} \] Label the origin \( O \), and assume that the \( m - 1 \) position vectors of the remaining points are linearly independent. Consider any partition of the \( m \) points into two subsets, \( S_1 \) and \( S_2 \), of order \( m_1 \) and \( m_2 \), so that \( m_1 + m_2 = m \). Let \( S_1 \) be the subset containing \( O \). Then the convex hull \( C_1 \) of \( S_1 \) is that set of points whose position vectors \( x \) satisfy

\[ x = \sum_{i=1}^{m_1} \alpha_i s_{1i}, \quad \sum_{i=1}^{m_1} \alpha_i = 1, \quad \alpha_i \geq 0 \tag{A.1} \]

where the \( s_{1i} \) are the position vectors of the \( m_1 \) points in \( S_1 \) (including the null position vector of the origin). Similarly, the convex hull \( C_2 \) of \( S_2 \) is that set of points whose position vectors \( x \) satisfy

\[ x = \sum_{i=1}^{m_2} \beta_i s_{2i}, \quad \sum_{i=1}^{m_2} \beta_i = 1, \quad \beta_i \geq 0 \tag{A.2} \]

where the \( s_{2i} \) are the position vectors of the \( m_2 \) points in \( S_2 \). Now suppose that \( C_1 \) and \( C_2 \) intersect. Then there exists an \( x \in \mathbb{R}^n \) which simultaneously satisfies Eq. (A.1) and Eq. (A.2). Subtracting these equations gives a linear combination of the \( m - 1 \) non-null position vectors which vanishes, which contradicts the assumption of linear independence. By the lemma, since \( C_1 \) and \( C_2 \) do not intersect, there exists a hyperplane separating \( S_1 \) and \( S_2 \). Since this is true for any choice of partition, the \( m \) points can be shattered.

It remains to show that if the \( m - 1 \) non-null position vectors are not linearly independent, then the \( m \) points cannot be shattered by oriented hyperplanes. If the \( m - 1 \) position vectors are not linearly independent, then there exist \( m - 1 \) numbers, \( \gamma_i \), such that

\[ \sum_{i=1}^{m-1} \gamma_i s_i = 0 \tag{A.3} \]

If all the \( \gamma_i \) are of the same sign, then we can scale them so that \( \gamma_i \in [0, 1] \) and \( \sum \gamma_i = 1 \). Eq. (A.3) then states that the origin lies in the convex hull of the remaining points; hence, by the lemma, the origin cannot be separated from the remaining points by a hyperplane, and the points cannot be shattered.

If the \( \gamma_i \) are not all of the same sign, place all the terms with negative \( \gamma_i \) on the right:

\[ \sum_{j \in I_1} |\gamma_j| s_j = \sum_{k \in I_2} |\gamma_k| s_k \tag{A.4} \]

where \( I_1, I_2 \) are the indices of the corresponding partition of \( S \setminus O \) (i.e. of the set \( S \) with the origin removed). Now scale this equation so that either \( \sum_{j \in I_1} |\gamma_j| = 1 \) and \( \sum_{k \in I_2} |\gamma_k| \leq 1 \); or \( \sum_{j \in I_1} |\gamma_j| \leq 1 \) and \( \sum_{k \in I_2} |\gamma_k| = 1 \). Suppose without loss of generality that the latter holds. Then the left hand side of Eq. (A.4) is the position vector of a point lying in the
convex hull of the points \( \{ \bigcup_{j \in I} s_j \} \cup O \) or, if the equality holds, of the points \( \{ \bigcup_{j \in I} s_j \} \), and the right hand side is the position vector of a point lying in the convex hull of the points \( \bigcup_{k \in J} s_k \), so the convex hulls overlap, and by the lemma, the two sets of points cannot be separated by a hyperplane. Thus the \( m \) points cannot be shattered. \( \blacksquare \)

**Theorem 4:** If the data is \( d \)-dimensional (i.e. \( L = \mathbb{R}^d \)), the dimension of the minimal embedding space, for homogeneous polynomial kernels of degree \( p \) \((K(x_1, x_2) = (x_1 \cdot x_2)^p, \ x_1, x_2 \in \mathbb{R}^d)\), is \((d+p-1)\).

**Proof:** First we show that the the number of components of \( \Phi(x) \) is \((p+d-1)\). Label the components of \( \Phi \) as in Eq. (79). Then a component is uniquely identified by the choice of the \( d \) integers \( r_i \geq 0, \sum_{i=1}^d r_i = p \). Now consider \( p \) objects distributed amongst \( d-1 \) partitions (numbered 1 through \( d-1 \)), such that objects are allowed to be to the left of all partitions, or to the right of all partitions. Suppose \( m \) objects fall between partitions \( q \) and \( q+1 \). Let this correspond to a term \( x_{q+1}^m \) in the product in Eq. (79). Similarly, \( m \) objects falling to the left of all partitions corresponds to a term \( x_1^m \), and \( m \) objects falling to the right of all partitions corresponds to a term \( x_d^m \). Thus the number of distinct terms of the form \( x_1^{r_1} x_2^{r_2} \cdots x_d^{r_d} \), \( \sum_{i=1}^d r_i = p \), \( r_i \geq 0 \) is the number of way of distributing the objects and partitions amongst themselves, modulo permutations of the partitions and permutations of the objects, which is \((p+d-1)\).

Next we must show that the set of vectors with components \( \Phi_{r_1, r_2 \ldots r_d}(x) \) span the space \( \mathcal{H} \). This follows from the fact that the components of \( \Phi(x) \) are linearly independent functions. For suppose instead that the image of \( \Phi \) acting on \( x \in \mathcal{L} \) is a subspace of \( \mathcal{H} \). Then there exists a fixed nonzero vector \( V \in \mathcal{H} \) such that

\[
\dim(\mathcal{H}) \sum_{i=1} V_i \Phi_i(x) = 0 \quad \forall x \in \mathcal{L}. \tag{A.5}
\]

Using the labeling introduced above, consider a particular component of \( \Phi \):

\[
\Phi_{r_1 r_2 \ldots r_d}(x), \quad \sum_{i=1}^d r_i = p. \tag{A.6}
\]

Since Eq. (A.5) holds for all \( x \), and since the mapping \( \Phi \) in Eq. (79) certainly has all derivatives defined, we can apply the operator

\[
(\frac{\partial}{\partial x_1})^{r_1} \cdots (\frac{\partial}{\partial x_d})^{r_d} \tag{A.7}
\]

to Eq. (A.5), which will pick that one term with corresponding powers of the \( x_i \) in Eq. (79), giving

\[
V_{r_1 r_2 \ldots r_d} = 0. \tag{A.8}
\]

Since this is true for all choices of \( r_1, \ldots, r_d \) such that \( \sum_{i=1}^d r_i = p \), every component of \( V \) must vanish. Hence the image of \( \Phi \) acting on \( x \in \mathcal{L} \) spans \( \mathcal{H} \). \( \blacksquare \)

**A.2. Gap Tolerant Classifiers and VC Bounds**

The following point is central to the argument. One normally thinks of a collection of points as being “shattered” by a set of functions, if for any choice of labels for the points, a function
from the set can be found which assigns those labels to the points. The VC dimension of that set of functions is then defined as the maximum number of points that can be so shattered. However, consider a slightly different definition. Let a set of points be shattered by a set of functions if for any choice of labels for the points, a function from the set can be found which assigns the incorrect labels to all the points. Again let the VC dimension of that set of functions be defined as the maximum number of points that can be so shattered.

It is in fact this second definition (which we adopt from here on) that enters the VC bound proofs (Vapnik, 1979; Devroye, Györfi and Lugosi, 1996). Of course for functions whose range is \( \{\pm 1\} \) (i.e. all data will be assigned either positive or negative class), the two definitions are the same. However, if all points falling in some region are simply deemed to be “errors”, or “correct”, the two definitions are different. As a concrete example, suppose we define “gap intolerant classifiers”, which are like gap tolerant classifiers, but which label all points lying in the margin or outside the sphere as errors. Consider again the situation in Figure 12, but assign positive class to all three points. Then a gap intolerant classifier with margin width greater than the ball diameter cannot shatter the points if we use the first definition of “shatter”, but can shatter the points if we use the second (correct) definition.

With this caveat in mind, we now outline how the VC bounds can apply to functions with range \( \{\pm 1, 0\} \), where the label 0 means that the point is labeled “correct.” (The bounds will also apply to functions where 0 is defined to mean “error”, but the corresponding VC dimension will be higher, weakening the bound, and in our case, making it useless). We will follow the notation of (Devroye, Györfi and Lugosi, 1996).

Consider points \( x \in \mathbb{R}^d \), and let \( p(x) \) denote a density on \( \mathbb{R}^d \). Let \( \phi \) be a function on \( \mathbb{R}^d \) with range \( \{\pm 1, 0\} \), and let \( \Phi \) be a set of such functions. Let each \( x \) have an associated label \( y_x \in \{\pm 1\} \). Let \( \{x_1, \cdots, x_n\} \) be any finite number of points in \( \mathbb{R}^d \); then we require \( \Phi \) to have the property that there exists at least one \( \phi \in \Phi \) such that \( \phi(x_i) \in \{\pm 1\} \) \( \forall \ x_i \). For given \( \phi \), define the set of points \( A \) by

\[
A = \{x : y_x = 1, \phi(x) = -1\} \cup \{x : y_x = -1, \phi(x) = 1\}
\] (A.9)

We require that the \( \phi \) be such that all sets \( A \) are measurable. Let \( \mathcal{A} \) denote the set of all \( A \).

**Definition:** Let \( x_i, i = 1, \cdots, n \) be \( n \) points. We define the empirical risk for the set \( \{x_i, \phi\} \) to be

\[
\nu_n(\{x_i, \phi\}) = \frac{1}{n} \sum_{i=1}^{n} I_{x_i \in A}
\] (A.10)

where \( I \) is the indicator function. Note that the empirical risk is zero if \( \phi(x_i) = 0 \ \forall \ x_i \).

**Definition:** We define the actual risk for the function \( \phi \) to be

\[
\nu(\phi) = P(x \in A).
\] (A.11)

Note also that those points \( x \) for which \( \phi(x) = 0 \) do not contribute to the actual risk.

**Definition:** For fixed \( (x_1, \cdots, x_n) \in \mathbb{R}^d \), let \( N_A \) be the number of different sets in

\[
\{(x_1, \cdots, x_n) \cap A : A \in \mathcal{A}\}
\] (A.12)
where the sets \( A \) are defined above. The \( n \)-th shatter coefficient of \( A \) is defined

\[
s(A, n) = \max_{x_1, \ldots, x_n \in \mathbb{R}^d} N_A(x_1, \ldots, x_n).
\]

We also define the VC dimension for the class \( A \) to be the maximum integer \( k \geq 1 \) for which \( s(A, k) = 2^k \).

**Theorem 8** (adapted from Devroye, Györfi and Lugosi, 1996, Theorem 12.6): Given \( \nu_n(\{x_i, \phi\}) \), \( \nu(\phi) \) and \( s(A, n) \) defined above, and given \( n \) points \( (x_1, \ldots, x_n) \in \mathbb{R}^d \), let \( \Phi' \) denote that subset of \( \Phi \) such that all \( \phi \in \Phi' \) satisfy \( \phi(x_i) \in \{ \pm 1 \} \) \( \forall x_i \). (This restriction may be viewed as part of the training algorithm). Then for any such \( \phi \),

\[
P(|\nu_n(\{x_i, \phi\}) - \nu(\phi)| > \epsilon) \leq 8s(A, n) \exp^{-n\epsilon^2/32}
\]

The proof is exactly that of (Devroye, Györfi and Lugosi, 1996), Sections 12.3, 12.4 and 12.5, Theorems 12.5 and 12.6. We have dropped the “sup” to emphasize that this holds for any of the functions \( \phi \). In particular, it holds for those \( \phi \) which minimize the empirical error and for which all training data take the values \( \{ \pm 1 \} \). Note however that the proof only holds for the second definition of shattering given above. Finally, note that the usual form of the VC bounds is easily derived from Eq. (A.14) by using \( s(A, n) \leq (en/h)^h \) (where \( h \) is the VC dimension) (Vapnik, 1995), setting \( \eta = 8s(A, n) \exp^{-n\epsilon^2/32} \), and solving for \( \epsilon \).

Clearly these results apply to our gap tolerant classifiers of Section 7.1. For them, a particular classifier \( \phi \in \Phi \) is specified by a set of parameters \( \{B, H, M\} \), where \( B \) is a ball in \( \mathbb{R}^d \), \( D \in \mathbb{R} \) is the diameter of \( B \), \( H \) is a \( d-1 \) dimensional oriented hyperplane in \( \mathbb{R}^d \), and \( M \in \mathbb{R} \) is a scalar which we have called the margin. \( H \) itself is specified by its normal (whose direction specifies which points \( H_+ \) (\( H_- \)) are labeled positive (negative) by the function), and by the minimal distance from \( H \) to the origin. For a given \( \phi \in \Phi \), the margin set \( S_M \) is defined as the set consisting of those points whose minimal distance to \( H \) is less than \( M/2 \). Define \( Z \equiv S_M \cap B \), \( Z_+ \equiv Z \cap H_+ \), and \( Z_- \equiv Z \cap H_- \). The function \( \phi \) is then defined as follows:

\[
\phi(x) = 1 \forall x \in Z_+ \quad \phi(x) = -1 \forall x \in Z_- \quad \phi(x) = 0 \text{ otherwise}
\]

and the corresponding sets \( A \) as in Eq. (A.9).

**Notes**

1. K. Müller, Private Communication

2. The reader in whom this elicits a sinking feeling is urged to study (Strang, 1986; Fletcher, 1987; Bishop, 1995). There is a simple geometrical interpretation of Lagrange multipliers: at a boundary corresponding to a single constraint, the gradient of the function being extremized must be parallel to the gradient of the function whose contours specify the boundary. At a boundary corresponding to the intersection of constraints, the gradient must be parallel to a linear combination (non-negative in the case of inequality constraints) of the gradients of the functions whose contours specify the boundary.

3. In this paper, the phrase “learning machine” will be used for any function estimation algorithm, “training” for the parameter estimation procedure, “testing” for the computation of the function value, and “performance” for the generalization accuracy (i.e. error rate as test set size tends to infinity), unless otherwise stated.
4. Given the name “test set,” perhaps we should also use “train set,” but the hobbyists got there first.
5. We use the term “oriented hyperplane” to emphasize that the mathematical object considered is the pair \{H, n\}, where H is the set of points which lie in the hyperplane and n is a particular choice for the unit normal. Thus \{H, n\} and \{H, -n\} are different oriented hyperplanes.
6. Such a set of m points (which span an \(m-1\) dimensional subspace of a linear space) are said to be in general position (Kolmogorov, 1970). The convex hull of a set of m points in general position defines an \(m-1\) dimensional simplex, the vertices of which are the points themselves.
7. The derivation of the bound assumes that the empirical risk converges uniformly to the actual risk as the number of training observations increases (Vapnik, 1979). A necessary and sufficient condition for this is that \(\lim_{m \to \infty} H(l)/l = 0\), where \(l\) is the number of training samples and \(H(l)\) is the VC entropy of the set of decision functions (Vapnik, 1979; Vapnik, 1995). For any set of functions with infinite VC dimension, the VC entropy is \(l \log 2\) hence these classifiers, the required uniform convergence does not hold, and so neither does the bound.
8. There is a nice geometric interpretation for the dual problem: it is basically finding the two closest points of convex hulls of the two sets. See (Bennett and Bredensteiner, 1998).
9. One can define the torque to be

\[
\Gamma_{\mu_1 \ldots \mu_{n-2}} = \epsilon_{\mu_1 \ldots \mu_n} \mu_{n-1} F_{\mu_n}
\]  

(A.16)

where repeated indices are summed over on the right hand side, and where \(\epsilon\) is the totally antisymmetric tensor with \(\epsilon_{1, \ldots n} = 1\). (Recall that Greek indices are used to denote tensor components). The sum of torques on the decision sheet is then:

\[
\sum_i \epsilon_{\mu_1 \ldots \mu_n} \mu_{n-1} F_{\mu_n} = \sum_i \epsilon_{\mu_1 \ldots \mu_n} \mu_{n-1} \alpha_i \gamma_{\mu_n} = \epsilon_{\mu_1 \ldots \mu_n} \mu_{n-1} \gamma_{\mu_n} = 0
\]  

(A.17)

10. In the original formulation (Vapnik, 1979) they were called “extreme vectors.”
11. By “decision function” we mean a function \(f(x)\) whose sign represents the class assigned to data point \(x\).
12. By “intrinsic dimension” we mean the number of parameters required to specify a point on the manifold.
13. Alternatively one can argue that, given the form of the solution, the possible \(w\) must lie in a subspace of dimension \(l\).
15. Thanks to A. Smola for pointing this out.
16. Many thanks to one of the reviewers for pointing this out.
17. The core quadratic optimizer is about 700 lines of C++. The higher level code (to handle caching of dot products, chunking, IO, etc) is quite complex and considerably larger.
18. Thanks to L. Kaufman for providing me with these results.
19. Recall that the “ceiling” sign \(\lceil \cdot \rceil\) means “smallest integer greater than or equal to.” Also, there is a typo in the actual formula given in (Vapnik, 1995), which I have corrected here.
20. Note, for example, that the distance between every pair of vertices of the symmetric simplex is the same: see Eq. (26). However, a rigorous proof is needed, and as far as I know is lacking.
21. Thanks to J. Shawe-Taylor for pointing this out.
22. V. Vapnik, Private Communication
23. There is an alternative bound one might use, namely that corresponding to the set of totally bounded non-negative functions (Equation (3.28) in (Vapnik, 1995)). However, for less functions taking the value zero or one, and if the empirical risk is zero, this bound is looser than that in Eq. (3) whenever \(\frac{H(l)(2/l+1)\log(\epsilon)/l}{4l} > 1/16\), which is the case here.
24. V. Blanz, Private Communication

References


