

Sparse Coding of Music Signals

Samer A. Abdallah* and Mark D. Plumbley†
Department of Electronic Engineering
King's College London

March 13, 2001

Abstract

We discuss the use of unsupervised learning techniques for the perception of music, focussing on the sparse coding of audio spectrograms. We show that a simple sparse coder can form the basis of a polyphonic transcription system, where each possible note induces a basis vector corresponding to the spectrum of that note. The outputs of the sparse coder then operate in parallel as note detectors. We introduce a new prior for approximating sparse distributions, and an optimisation algorithm for dealing with sharply-peaked priors.

1 Unsupervised Learning and Perception

For some time now, unsupervised learning has appeared to be a most effective framework for understanding at least the early stages of sensory processing (see, for example [Barlow, 1989, Field, 1994]). Perceptual processes are thought to be driven by a need to find representations of sensory signals that are both efficient and useful, as quantified by the related measures of low redundancy, high information content, and statistical independence [Atick, 1992]. This statistical independence is linked with the idea that one of the goals of perception may be to infer the *independent causes* of the signals, which causes constitute an explanation of sorts, and by which we may be said to *understand* what we sense.

More recently, Independent Component Analysis (ICA) [Hyvärinen, b] and the related technique of *sparse coding* using an overcomplete basis [Field and Olshausen, 1996] have emerged as promising avenues of research, and have been shown to fit into a wider class of statistical methods based on generative modelling and maximum-likelihood inference [Cardoso, 1997, Olshausen, 1996].

Our work is directed towards adapting and applying these ideas in the area of auditory perception, specifically for the perception of music. Our initial aim is to use sparse coding to analyse musical objects such as notes and percussive events, identifying both the pitch and the instrument.

*e-mail: samer.abdallah@kcl.ac.uk

†e-mail: mark.plumbley@kcl.ac.uk

2 Working with Audio Spectra

Much of this work was prompted by the observation that spectrograms of polyphonic music (see Appendix A.1 for computational details) are rather reminiscent of the ‘bars’ problem [Földiák, 1990] often used to test factorial coding algorithms. This involves forming each input pattern by super-imposing a small number of patterns chosen at random from a larger set or dictionary of basic patterns. In a spectrogram, each note appears as a pattern of harmonics (see figure 1). There are a large number of possible notes, but in general, only a few are present at a time. If these notes come and go more or less independently, then it should be possible for something like an ICA algorithm to learn the individual notes and thus be able to decompose a spectrum into its constituent parts.

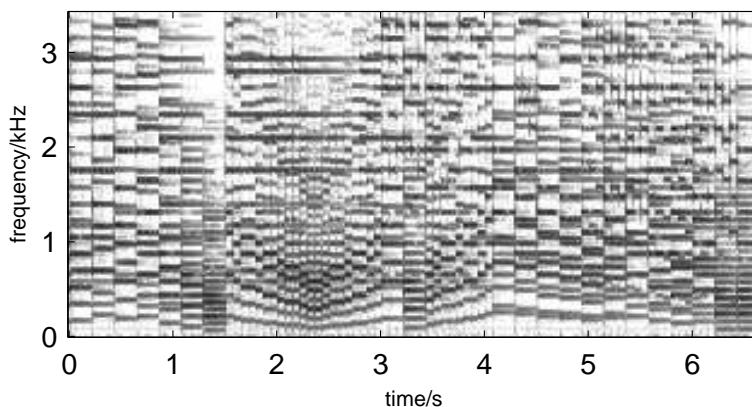


Figure 1: A music spectrogram: part of Bach’s *Partita in A minor* for keyboard, BWV827, played on a synthesised harpsichord from a MIDI recording.

There are several reasons why a spectrogram might not be the ideal input to a sparse coder, but it does have some redeeming features (not the least of which for practical purposes is computational efficiency). Some of the factors to consider are:

Limited resolution It is well known that spectrograms have limited simultaneous resolution in time and frequency [Gabor, 1947, Flandrin and Rioul, 1990], with frequency resolution bought at the expense of time resolution and *vice versa*. This means that time-frequency features are not as distinct as they might otherwise be.

Irreversibility It is not possible to get perfect reconstruction from a spectrogram because phase information is irretrievably lost.

Noise statistics The algorithms to be discussed in Section 3 assume Gaussian noise statistics, but Gaussian noise in the signal does not appear as Gaussian noise in a power spectrogram. In absence of any other signal, it actually appears as random noise with a (single sided) *exponential* distribution (*i.e.* of the form $p(u) = \lambda e^{-\lambda u}, u \geq 0$). If, however, we use the

magnitude spectrogram, by taking the square root of the power spectrogram, we get a distribution of the form $p(v) = 2\lambda v e^{-\lambda v^2}$, where $v = \sqrt{u}$. (See Appendix A.2 for details.)

Linearity or otherwise of mixing The sparse coders to be discussed in Section 3 assumed a linear generative model, which means that they will be theoretically capable of disentangling the independent causes of their input *if* these causes combine additively. Now, power spectrograms *do* add linearly if the phases of the sources are uncorrelated, but we do not expect this to be the case for musical instruments that are in tune with each other. If harmonics from two instruments overlap (which they will if the notes are consonant) then we may get constructive, destructive or beating interference, which effects are linear in the time domain (and in the linear Fourier transform) but non-linear in the spectrogram.

In addition, we do not expect to be able to discover the independent causes that go into the production of a *single* instrumental sound since these will probably combine in some more complicated fashion. For example, if we consider loudness, pitch and spectral envelope to be independent causes, they would combine multiplicatively in the spectral domain.

Phase invariance Though the human auditory system is sensitive to phase, phase is largely redundant as far as *musical* content is concerned. Melody and rhythm are both invariant to phase changes and timbre is only slightly affected (see for example, [Plomp, 1976]). The kind of information that is present in the phases is more to do with spatialisation and the acoustic environment. Put simply, because it is so strongly affected by the environment, phase is not a *reliable* source of information about the source. The phase invariance of the spectrogram is therefore an attractive feature.

At any rate, this is not meant to be an advocacy of spectrograms—we are more interested in the adaptive algorithms that will operate on them. There are other time-frequency representations that are currently receiving attention in the music processing community, such as scalograms and correlograms [Flandrin and Rioul, 1990, Duda et al., 1990]; one of the aims of our work is to discover what it is about these representations that makes them useful, and to develop adaptive processing strategies that incorporate those useful properties and are not dependent on a particular choice of representation. (See [Abdallah and Plumbley, 1999] for a discussion of the philosophy behind this.) We have been using spectrograms only because they form a convenient first base on which to develop those strategies.

3 Sparse Coding

Although it is difficult to formulate a rigorous definition, the general consensus is that a sparse code is one which represents information “in terms of a small number of descriptors out of a large set” [Field and Olshausen, 1996], that is, “only a fraction of the code elements are actively used to represent a typical pattern” [Harpur, 1997].

In numeric terms, this is often taken [Harpur, 1997, Hyvärinen, 1998] to mean that most of the elements are zero or practically zero most of the

time. The difficulty lies in defining what ‘practically zero’ means. The implicit assumption is that those values which are ‘close to zero’ may be treated as being exactly zero with little or no loss of useful information.

With reference to the marginal distributions of the code elements, several workers [Olshausen, 1996, Lewicki and Sejnowski, 1998, Hyvärinen, 1998] go on to equate sparsity with high *kurtosis*. The kurtosis of a random variable is defined as $\kappa_4(x) = \langle x^4 \rangle - 3\langle x^2 \rangle^2$, and is zero for Gaussians, positive for distributions with a more concentrated peak at zero and heavier tails, and negative for those with a broader central peak and lighter tails. However, as Harpur states [Harpur, 1997], “high kurtosis is a good indicator of high sparseness and low entropy *in the unimodal case*. Its usefulness in more general cases is less clear.” In addition, such measures based on high order cumulants are more susceptible to outliers, and estimates of kurtosis using a finite sample size become less and less reliable the heavier the tails of the distribution. In fact, for distributions with tails that decay more slowly than x^{-5} , the kurtosis is not even defined, because the integral for $\langle x^4 \rangle$ does not converge.

We would suggest that kurtosis is not a good way to characterise sparsity, and that it is better to think about active and inactive elements. That small values should be admissible as ‘practically zero’, (*i.e.* inactive) is, in our opinion, merely a concession to the fact that distributions with infinitely concentrated peaks at zero are difficult to deal with in the current framework. Bearing this in mind, we propose the following qualitative definition of sparsity: namely, that the distribution is *strongly and tightly peaked at zero*, by which we mean:

1. the peak contains much of the ‘mass’ of the distribution;
2. the peak is narrow in relation to the overall width of the distribution, as characterised by *some appropriate measure*, the implication being that any value smaller than the width of the peak may be set to zero without significant loss of information.

Note that an ‘appropriate’ width measure need not be variance, which may be poorly defined for some distributions. It could, for example, be a mean absolute value, a maximum value, or a percentile.

Given that a distribution satisfies these criteria, then a useful measure of sparsity is simply the probability mass of the peak itself, that is, the probability that a given element is inactive.

3.1 A Generative Model

This remainder of this section contains an overview of some sparse coding algorithms. They are all based on a linear generative model where the observed data vectors, $\mathbf{x} \in \mathbb{R}^L$ are derived from a number of hidden sources, s_i , arranged into a state vector $\mathbf{s} \in \mathbb{R}^M$, according to

$$\mathbf{x} = \mathbf{A}\mathbf{s} + \boldsymbol{\epsilon}, \tag{1}$$

where \mathbf{A} is an $L \times M$ basis matrix and $\boldsymbol{\epsilon}$ is a Gaussian random vector of covariance $\langle \boldsymbol{\epsilon}\boldsymbol{\epsilon}^T \rangle = \boldsymbol{\Lambda}_\epsilon^{-1}$. This gives us, putting $\boldsymbol{\epsilon} = \mathbf{x} - \mathbf{A}\mathbf{s}$, the probability of observing a particular \mathbf{x} given a known basis \mathbf{A} and sources \mathbf{s} :

$$P(\mathbf{x}|\mathbf{A}, \mathbf{s}) = N \exp -\frac{1}{2}\boldsymbol{\epsilon}^T \boldsymbol{\Lambda}_\epsilon \boldsymbol{\epsilon}, \tag{2}$$

where N is a normalisation constant. In practice, we will usually set $\mathbf{\Lambda}_\epsilon = \sigma^{-2}\mathbf{I}$, that is, a scalar multiple of the identity matrix, corresponding to independent Gaussian noise of variance σ^2 on each input component x_i , otherwise known as *spherical Gaussian noise*. The elements of the state vector \mathbf{s} are assumed to be independent and drawn from a sparse distribution, $p(s)$, giving

$$P(\mathbf{s}) = \prod_{i=1}^M p(s_i). \quad (3)$$

The goal is to find the most probable values of both the basis matrix \mathbf{A} and the sources \mathbf{s} from observations of \mathbf{x} . In fact, as is common practice with this sort of problem [Cardoso, 1997, MacKay, 1996], we will determine the matrix \mathbf{A} that maximises the likelihood of observing the data \mathbf{x} , by marginalising out \mathbf{s} and maximising $P(\mathbf{x}|\mathbf{A})$. Estimates of \mathbf{s} will be made given our current estimate of \mathbf{A} , by maximising $P(\mathbf{s}|\mathbf{A}, \mathbf{x})$. The first process is usually called *learning*, the second, *inference*.

3.2 Inference and the Activation Dynamics

If \mathbf{A} is known, we can estimate \mathbf{s} given \mathbf{x} by considering the posterior distribution

$$P(\mathbf{s}|\mathbf{A}, \mathbf{x}) = \frac{P(\mathbf{x}|\mathbf{A}, \mathbf{s})P(\mathbf{s})}{P(\mathbf{x}|\mathbf{A})}. \quad (4)$$

A maximum *a posteriori* (MAP) estimate is given by

$$\hat{\mathbf{s}} = \arg \max_{\mathbf{s}} [\log P(\mathbf{x}|\mathbf{A}, \mathbf{s}) + \log P(\mathbf{s})] \quad (5)$$

If the posterior is sufficiently smooth, we can find this maximum by gradient ascent or some other gradient-based optimization procedure, the conditions for zeroing the gradient being

$$\frac{\partial \log P(\mathbf{x}|\mathbf{A}, \mathbf{s})}{\partial s_i} + \frac{\partial \log P(\mathbf{s})}{\partial s_i} = 0 \quad \forall 1 \leq i \leq M. \quad (6)$$

It will be useful to define the scalar function $\gamma(s)$ and the vector-valued function $\boldsymbol{\gamma}(\mathbf{s})$ as follows:

$$\gamma(s) = -\frac{d}{ds} \log p(s) \quad (7)$$

$$[\boldsymbol{\gamma}(\mathbf{s})]_i = \gamma(s_i), \quad (8)$$

which implies that $\boldsymbol{\gamma}(\mathbf{s}) = -\nabla \log P(\mathbf{s})$. Using this and the expression for $P(\mathbf{x}|\mathbf{A}, \mathbf{s})$ from equation 2, we can write the zero-gradient condition as

$$\mathbf{A}^T \mathbf{\Lambda}_\epsilon (\mathbf{x} - \mathbf{A}\mathbf{s}) - \boldsymbol{\gamma}(\mathbf{s}) = 0. \quad (9)$$

Expressed in terms of a steepest-ascent gradient optimisation, local maxima of the posterior can be found as stable fixed points of

$$\frac{d\mathbf{s}}{dt} = \mathbf{A}^T \mathbf{\Lambda}_\epsilon \boldsymbol{\epsilon} - \boldsymbol{\gamma}(\mathbf{s}), \quad (10)$$

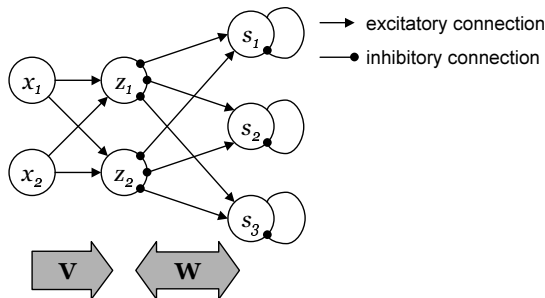


Figure 2: A neural implementation of the activation dynamics.

though of course we cannot guarantee that this will be the global maximum unless the posterior is unimodal. These activation dynamics can be interpreted as an *error correcting* mechanism, where the reconstruction error $\epsilon = \mathbf{x} - \mathbf{A}\mathbf{s}$ drives changes in the sources \mathbf{s} , but where each element of \mathbf{s} is subject to non-linear decay process determined by the function $\gamma(\cdot)$.

The computation can be implemented locally in a three-layer neural network, illustrated in figure 3.2. The units of first layer are clamped to the input \mathbf{x} . The second layer receives feed-forward input from the first layer and feed-back from the third, with activities given by

$$z_k = \sum_j V_{kj} x_j - \sum_i W_{ki} s_i,$$

where the matrices V_{kj} and W_{ki} represent the connection strengths between the layers. The units of the third layer, in addition to receiving feed-forward input from the second, each have non-linear self-inhibitory connections, implementing the effect of the term $\gamma(\mathbf{s})$ in equation 10, and are governed by the *dynamic* equation,

$$\frac{ds_i}{dt} = \sum_k z_k W_{ki} - \gamma(s_i).$$

The matrices \mathbf{V} and \mathbf{W} are given by

$$\mathbf{V} = \mathbf{\Lambda}_\epsilon^{\frac{1}{2}}, \quad \mathbf{W} = \mathbf{V}\mathbf{A}.$$

The matrix square-root is guaranteed to exist because the covariance of the noise (and hence $\mathbf{\Lambda}_\epsilon$) is positive semi-definite. If we choose a *symmetric* square-root, so that $\mathbf{V}^T \mathbf{V} = \mathbf{\Lambda}_\epsilon$, then we will get the desired dynamics at the output layer.

The significance of the connection weights \mathbf{V} from the first to the second layer is that they serve to whiten or decorrelate not the data, as is common in ICA algorithms [Karhunen,], but the *noise*. That is, regardless of the noise covariance structure on the input, it appears as decorrelated noise of unit variance at the middle layer. If the noise is decorrelated to begin with, so that $\mathbf{\Lambda}_\epsilon$ is diagonal, then so much the better: \mathbf{V} will also be diagonal. The computation of $\hat{\mathbf{s}}$ is picked up again in Section 5.

3.3 Learning

To estimate the value of \mathbf{A} , we maximise the following objective function:

$$\mathcal{L} = \langle \log P(\mathbf{x}|\mathbf{A}) \rangle_{P(\mathbf{x})}, \quad (11)$$

where the angle brackets $\langle \dots \rangle$ denote an expectation taken over the distribution of the observed vectors \mathbf{x} . The model distribution or data likelihood is given by

$$P(\mathbf{x}|\mathbf{A}) = \int_{\mathbb{R}^M} ds P(\mathbf{x}|\mathbf{A}, \mathbf{s})P(\mathbf{s}), \quad (12)$$

where this and all the following integrals are over the M -dimensional space available to \mathbf{s} . We can gain an insight into how the learning rule is constructed by casting this into the language of statistical physics: defining the energy as

$$\mathcal{E}(\mathbf{s}) = -\log P(\mathbf{x}|\mathbf{A}, \mathbf{s})P(\mathbf{s}), \quad (13)$$

we can write the posterior distribution over \mathbf{s} as

$$P(\mathbf{s}|\mathbf{A}, \mathbf{x}) = \frac{e^{-\mathcal{E}(\mathbf{s})}}{\mathcal{Z}},$$

where the partition function, \mathcal{Z} , is defined as

$$\mathcal{Z} = \int e^{-\mathcal{E}(\mathbf{s})} d\mathbf{s} = P(\mathbf{x}|\mathbf{A}). \quad (14)$$

Thus, maximising the posterior is equivalent to minimising the energy: $\hat{\mathbf{s}} = \text{argmin}_{\mathbf{s}} \mathcal{E}(\mathbf{s})$. We intend to maximise \mathcal{L} by gradient ascent. Letting θ stand for some scalar parameter we wish to optimise (*e.g.* an element of \mathbf{A}), this would require, for a continuous time gradient ascent, that we set

$$\frac{d\theta}{dt} = \frac{\partial \mathcal{L}}{\partial \theta} = \left\langle \frac{\partial \log \mathcal{Z}}{\partial \theta} \right\rangle_{P(\mathbf{x})}.$$

If, instead, we wish to perform on-line learning using a discrete time stochastic gradient ascent, then instead of averaging over the distribution of \mathbf{x} , we sample from that distribution, giving an update rule for each presentation of \mathbf{x} :

$$\Delta\theta = \eta \frac{\partial \log \mathcal{Z}}{\partial \theta},$$

where η is a learning rate parameter. Using equation 3.3, we find that

$$\begin{aligned} \frac{\partial \log \mathcal{Z}}{\partial \theta} &= \frac{1}{\mathcal{Z}} \int \frac{\partial}{\partial \theta} e^{-\mathcal{E}(\mathbf{s})} d\mathbf{s} \\ &= -\frac{1}{\mathcal{Z}} \int e^{-\mathcal{E}(\mathbf{s})} \frac{\partial \mathcal{E}(\mathbf{s})}{\partial \theta} d\mathbf{s} \\ &= -\int d\mathbf{s} P(\mathbf{s}|\mathbf{A}, \mathbf{x}) \frac{\partial \mathcal{E}(\mathbf{s})}{\partial \theta}, \end{aligned}$$

which has the form of an expectation over the posterior distribution. Using equations 2 and 13 we obtain

$$\frac{\partial \mathcal{E}(\mathbf{s})}{\partial A_{ij}} = -[\mathbf{\Lambda}_\epsilon(\mathbf{x} - \mathbf{A}\mathbf{s})\mathbf{s}^T]_{ij}, \quad (15)$$

so the update rule for the basis matrix \mathbf{A} is

$$\begin{aligned}\Delta\mathbf{A} &= \eta\mathbf{\Lambda}_\epsilon \int (\mathbf{x} - \mathbf{A}\mathbf{s})\mathbf{s}^T P(\mathbf{s}|\mathbf{A}, \mathbf{x})d\mathbf{s} \\ &= \eta\mathbf{\Lambda}_\epsilon \langle \epsilon\mathbf{s}^T \rangle_{P(\mathbf{s}|\mathbf{x}, \mathbf{A})}.\end{aligned}\tag{16}$$

As in the activation dynamics, the learning dynamics involve the reconstruction error $\epsilon = \mathbf{x} - \mathbf{A}\mathbf{s}$.

3.4 Approximations to Exact Learning

This is an appropriate point to describe the sparse coding algorithms of Field and Olshausen [Field and Olshausen, 1996], Harpur [Harpur, 1997], and Lewicki and Sejnowski [Lewicki and Sejnowski, 1998], to see how they relate to equation 16. The integration over the posterior in equation 16 grows exponentially as the dimensionality of \mathbf{s} increases. Where the algorithms differ is in the approximation used to make this integral tractable.

3.4.1 Delta Approximation

Both Field and Olshausen’s sparse coder [Field and Olshausen, 1996] and Harpur’s Recurrent Error Correction (REC) network [Harpur, 1997] can be derived by collapsing the posterior $P(\mathbf{s}|\mathbf{A}, \mathbf{x})$ to a delta distribution at its maximum, $\hat{\mathbf{s}}$, as determined previously according to the methods of Section 3.2. Setting

$$P(\mathbf{s}|\mathbf{A}, \mathbf{x}) = \delta(\mathbf{s} - \hat{\mathbf{s}}),$$

in equation 16 (where $\delta(\cdot)$ denotes in this case an M -dimensional delta distribution) leads to a weight update of

$$\Delta\mathbf{A}_{\text{REC}} = \eta\mathbf{\Lambda}_\epsilon \hat{\epsilon}\hat{\mathbf{s}}^T,\tag{17}$$

where we have defined $\hat{\epsilon} = \mathbf{x} - \mathbf{A}\hat{\mathbf{s}}$, and which, if the matrix $\mathbf{\Lambda}_\epsilon$ is diagonal, can be interpreted as Hebbian learning between the sources and the reconstruction error. Left to itself, this would result in the basis matrix growing without limit and the estimated sources $\hat{\mathbf{s}}$ tending steadily to zero. An explicit normalisation step is required to keep this from happening.

3.4.2 Gaussian Approximation

Lewicki and Sejnowski [Lewicki and Sejnowski, 1998] use a multivariate Gaussian approximation to the posterior around its maximum at $\hat{\mathbf{s}}$, setting

$$P(\mathbf{s}|\mathbf{A}, \mathbf{x}) \approx \sqrt{\frac{|\mathbf{H}|}{(2\pi)^M}} \exp\left[-\frac{1}{2}(\mathbf{s} - \hat{\mathbf{s}})^T \mathbf{H}(\mathbf{s} - \hat{\mathbf{s}})\right].$$

By construction, the mean of the Gaussian is $\hat{\mathbf{s}}$, and its covariance matrix is \mathbf{H}^{-1} , where \mathbf{H} is the Hessian of the (exact) *log*-posterior evaluated at $\hat{\mathbf{s}}$:

$$\mathbf{H} = -\nabla\nabla^T \log P(\mathbf{s}|\mathbf{A}, \mathbf{x}) = \nabla\nabla^T \mathcal{E}(\mathbf{s}).$$

This eventually results (see their original paper for details) in a weight update of the form

$$\Delta\mathbf{A}_{\text{Gauss}} = \eta\mathbf{\Lambda}_\epsilon (\hat{\epsilon}\hat{\mathbf{s}}^T - \mathbf{A}\mathbf{H}^{-1}).\tag{18}$$

By taking into account the width of the posterior around its peak, the update rule gains the decay term $\mathbf{A}\mathbf{H}^{-1}$, which solves the problem of unlimited weight growth. The basis vectors are automatically normalised so that the widths of the marginal distributions of the outputs match that of the prior.

The computation of \mathbf{H}^{-1} cannot be implemented simply in a neural architecture; Lewicki and Sejnowski actually employ a more useful form given by

$$\Delta\mathbf{A}_{LS} = \eta\mathbf{A}[\gamma(\hat{\mathbf{s}})\hat{\mathbf{s}}^T - \mathbf{I}], \quad (19)$$

which can be obtained from equation 18 in the following way: Firstly, following MacKay [MacKay, 1996], the right hand side of equation 18 is pre-multiplied by the positive-definite matrix $\mathbf{A}\mathbf{A}^T$, to give a *covariant* update rule. MacKay’s paper contains a fuller discussion, but loosely speaking, it results in an algorithm which is dimensionally consistent (in terms of the units used to measure the quantities involved) and *scale invariant* with respect to \mathbf{A} . This is a good thing because it yields an approximation to the *natural gradient* [ichi Amari, 1998].

Secondly, according to equation 9, at the maximum of the posterior we should have $\mathbf{A}^T\mathbf{\Lambda}_\epsilon\hat{\boldsymbol{\epsilon}} = \gamma(\hat{\mathbf{s}})$. Lastly, a further approximation can be made that results in $\mathbf{A}^T\mathbf{\Lambda}_\epsilon\mathbf{A}\mathbf{H}^{-1} \approx \mathbf{I}$. Taken together, these steps yield

$$\begin{aligned} \mathbf{A}\mathbf{A}^T \cdot \Delta\mathbf{A}_{\text{Gauss}} &= \mathbf{A}\mathbf{A}^T \cdot \eta\mathbf{\Lambda}_\epsilon(\hat{\boldsymbol{\epsilon}}\hat{\boldsymbol{\epsilon}}^T - \mathbf{A}\mathbf{H}^{-1}) \\ &= \eta\mathbf{A}(\mathbf{A}^T\mathbf{\Lambda}_\epsilon\hat{\boldsymbol{\epsilon}}\hat{\boldsymbol{\epsilon}}^T - \mathbf{A}^T\mathbf{\Lambda}_\epsilon\mathbf{A}\mathbf{H}^{-1}) \\ &\approx \eta\mathbf{A}[\gamma(\hat{\mathbf{s}})\hat{\mathbf{s}}^T - \mathbf{I}], \end{aligned}$$

as stated in equation 19, and which can be easily be implemented neurally.

4 The Form of the Prior

Even before doing any processing, the energy in a spectrogram is already extremely sparsely distributed, in the sense that most of the spectrogram consists of a low level background with relatively small, concentrated regions of high energy density. We can see this in the marginal distributions of individual spectral bands (see figure 3 and also the discussion of marginal distributions in [Jordanov and Penev, 1999]). The distributions are strongly peaked toward zero, with tails that decay much more slowly than a Gaussian (which would appear as a parabola curving downwards). In fact, in many cases, they decay more slowly than a Laplacian (which would appear as a straight line of negative gradient).

However, there *are* redundancies present in spectrograms, as evidenced by the fact that the activity tends to be localised in recognisable structures in the time-frequency plane. Bearing in mind the initial sparsity of the spectrogram, we would expect the result of sparse coding to be a yet sparser representation. Hence, we should investigate how a sparse coder would cope with an extremely sparse prior, $p(s)$. A number of observations can be made:

- A Laplacian is the sparsest prior we can have without inducing a potentially multimodal posterior. We can see why this is so by considering the energy function described in equation 13:

$$\mathcal{E}(\mathbf{s}) = -\log P(\mathbf{x}|\mathbf{A}, \mathbf{s}) - \log P(\mathbf{s}).$$

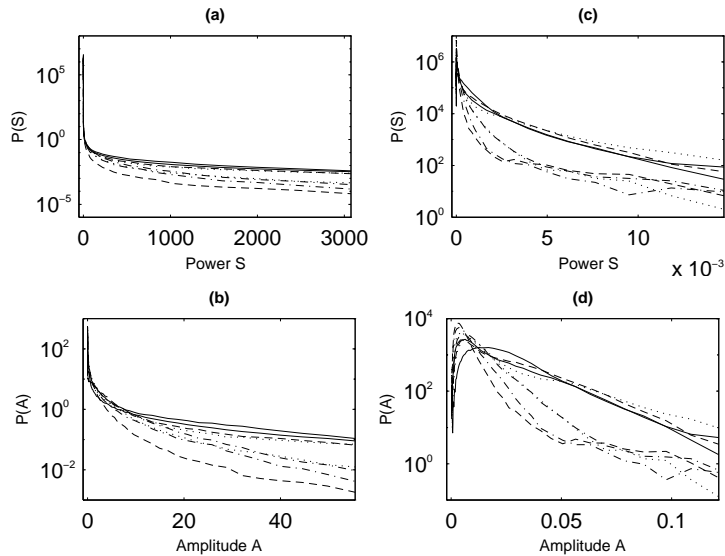


Figure 3: Marginal densities of spectral bands. Each line type indicates the distribution (over time) of activity at a particular frequency. (a) and (b) were taken directly from power (S) and magnitude (A) spectrograms respectively (with $A = \sqrt{S}$), whereas in (c) and (d), the audio signal was fed through a long time-constant automatic gain control, to remove large (time) scale fluctuations in loudness. Even with these variations removed, the densities can still be considered to be quite sparse (see text). This particular signal was taken from an ordinary audio CD (the Penguin Cafe Orchestra’s *Signs of Life*), but qualitatively similar results were obtained from a number of CDs representing a range of musical styles.

Finding the minimum of this is equivalent to finding the maximum of the posterior $P(\mathbf{s}|\mathbf{A}, \mathbf{x})$. Equation 2 implies that $-\log P(\mathbf{x}|\mathbf{A}, \mathbf{s})$ is a quadratic form with one global minimum, and second derivatives that are everywhere positive (its Hessian is positive definite everywhere). The only way for the energy to have non-global minima is for $-\log P(\mathbf{s})$ to have a Hessian which is not everywhere positive definite. Since $\log P(\mathbf{s}) = \sum_i \log p(s_i)$, this implies that $-\log p(s)$ must have a negative second derivative for some value of s . An alternative way of stating this is that if $-\log p(s)$ is *convex*, there will be only one global minimum of the energy. Now, because a Laplacian prior has a log-prior made of two linear segments ($-\log p(s) = |s|$), it is on the verge of being non-convex. If the distribution were to become any ‘peakier’, the log-prior would cease to be convex, and we could no longer guarantee a unimodal posterior.

As soon as the posterior becomes multimodal, not only does the global maximum become harder to find using a local gradient algorithm, it becomes less representative of the distribution as a whole, and less useful for approximating the expectation in equation 16. In essence, the various approximations described in Section 3.3 break down.

- A gradient discontinuity at zero (as in the Laplacian) is desirable, because it gives a code with many *exact* zeros rather than just very small values

(see Figure 4(c) and the discussion of Section 5). This is closer to our intuitive concept sparsity and is more attractive than the alternative of thresholding small values, because it does not involve an arbitrary choice of threshold. Also, thresholding introduces irreversible reconstruction errors, whereas if the zeros come about because of the shape of the prior, there is a chance for the remaining active units to compensate via the ‘error-correcting’ action of the activation dynamics described in equation 10.

- One way of looking at a very sparse random source is to view it as a mixture of zeros and some continuous random variable like a Gaussian or Laplacian. This lends a certain binary flavour to our sources: they are either exactly off or on with some non-zero value, which is an appealing quality for musical notes—yes, notes have loudness, but much of the time we are simply interested in whether or not a note is there.

The resultant probability density of such a ‘sparsified’ variable will have a delta distribution at zero. The posterior will potentially be multimodal and will have an infinite peaks wherever any of the components of \mathbf{s} is zero. Needless to say, this will pose problems for any procedure based on MAP estimation, which will always yield $\hat{\mathbf{s}} = \mathbf{0}$. This is discussed further in Section 9.

In order to avoid dealing with such a mixed discrete/continuous prior, and to allow us to use the relatively simple algorithms described in Section 3, we constructed a continuous approximation to a sparsified Laplacian, to be described in the next section.

5 An Approximate Sparsified Laplacian Prior

Our approximation to a ‘sparsified’ Laplacian consists of two Laplacian pieces, the central part forming a narrow peak, as illustrated in figure 4(a). The parameters μ and K control the width and relative mass of the central peak:

$$p(s) = \begin{cases} Ne^{-|s|} & : |s| \geq \mu, \\ NCe^{-K|s|} & : |s| < \mu, \end{cases} \quad (20)$$

where N is a normalisation constant, and $C = e^{\mu(K-1)}$ to ensure continuity. The width parameter μ is intended to be small, in keeping with our working definition of sparsity outlined in Section 3, in which case the distribution’s sparsity can be measured as

$$\Pr(|s| < \mu) = \frac{e^{\mu K} - 1}{e^{\mu K} + K - 1},$$

allowing us to calculate the parameters required for a given level of sparsity. The associated function $\gamma(s)$, illustrated in figure 4(b), is given by

$$\gamma(s) = \begin{cases} \text{sgn } s & : |s| \geq \mu, \\ K \text{sgn } s & : |s| < \mu, \end{cases} \quad (21)$$

This prior results in a thresholding behaviour when computing \hat{s} . Following [Hyvärinen, 1998], we consider the one-dimensional case ($N = M = 1$), in which

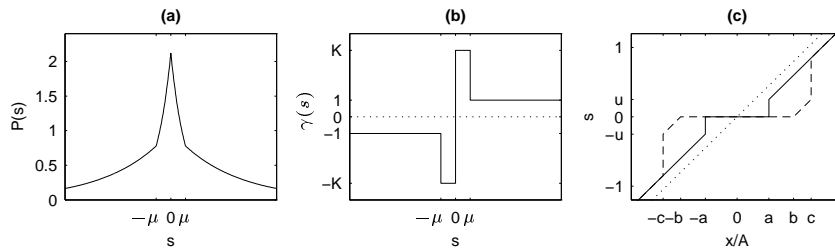


Figure 4: Approximation to ‘sparsified’ Laplacian, constructed piecewise from exponential segments. (a) Prior, (b) $\gamma(s)$ and (c) shrinkage operator. Where the shrinkage operator is multi-valued, the actual value reached depends on the details of the optimisation procedure used to find \hat{s} . The two options are shown as the solid and dashed lines; the dotted line shows the line $\hat{s} = x/A$.

case, the stationarity condition in equation 9 gives, at the maximum of the posterior,

$$\frac{1}{\sigma^2} A(x - A\hat{s}) = \gamma(\hat{s}) \quad (22)$$

which we can rewrite as

$$x/A = g(\hat{s}) = \hat{s} + \frac{\sigma^2}{A^2} \gamma(\hat{s}).$$

If the function $g(\cdot)$ was invertible, we would be able to obtain \hat{s} as a function of x : $\hat{s} = g^{-1}(x/A)$. In our case, $g(\cdot)$ is not invertible, but we can construct a shrinkage operator—see figure 4(c)—which, though not fully determined by equation 22, yields a value of \hat{s} that satisfies the stationarity condition. Where the posterior is bimodal, there are two branches in the graph, representing the two local maxima. Strictly speaking, we should compare their heights to find the global maximum, though in practice, a gradient based optimiser may find one or the other depending on its initial conditions.

6 The Activation Algorithm

To find the maximum of the posterior (or equivalently, the minimum of the energy function $\mathcal{E}(s)$ in Equation 13) we need an optimisation algorithm. Practical methods work best with functions that are approximately quadratic near the minimum, but our prior has a gradient discontinuity at zero, inducing corresponding discontinuities in the energy at coordinate zeros. If the minimum of the energy function occurs at one of these creases, a gradient based optimisation procedure will have problems converging.

To address this, we implemented a modified quasi-Newton optimiser which explicitly recognises the fact that there may be gradient discontinuities at component zeros of the vector being optimised. In a fashion similar to Endres and Földiák’s quadratic programming method [Endres and Földiák, 1999], our algorithm maintains a set of active dimensions, and could be called an *active set quasi-Newton* optimiser. In fact, the same modification can be applied to any iterative gradient-based optimiser that involves a step length computation; for example, we also produced a conjugate gradient version.

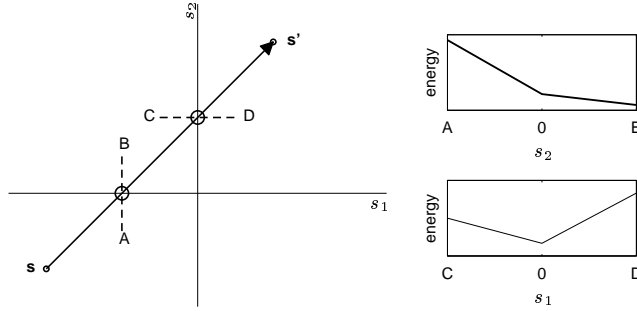


Figure 5: A Two dimensional illustration of the operation of the modified active set optimiser. Given the proposed step from \mathbf{s} to \mathbf{s}' , and the local behaviour of the gradient along the segments AB and CD , the modified optimiser would truncate the step at the *second* zero crossing, and inactivate the *first* dimension, s_1 .

The optimisation is an iterative procedure, involving a current point $\mathbf{s} \in \mathbb{R}^M$ and two sets I_0 and I_1 which contain the indices of the inactive and active elements (or dimensions) of \mathbf{s} respectively:

$$I_0 \cap I_1 = \emptyset, \quad I_0 \cup I_1 = \{1, \dots, M\}.$$

Inactive dimensions are set to zero and do not take part in the current optimisation step, though they may subsequently be activated:

$$i \in I_0 \implies s_i = 0$$

To determine whether or not dimension i should be active, we define a boolean indicator function $Q(\mathbf{s}, i)$, which, assuming that $s_i = 0$, depends on the signs of the gradient of the cost function on either side of the point $s_i = 0$. These gradients are given by

$$z_+(\mathbf{s}, i) = \lim_{s_i \downarrow 0^+} \frac{\partial \mathcal{E}(\mathbf{s})}{\partial s_i}, \quad z_-(\mathbf{s}, i) = \lim_{s_i \uparrow 0^-} \frac{\partial \mathcal{E}(\mathbf{s})}{\partial s_i}, \quad (23)$$

where the limits are taken tending *down* to a value just above zero, and *up* to a value just below zero respectively. The indicator function is given by

$$Q(\mathbf{s}, i) = \begin{cases} 1 & : [\text{sgn } z_+(\mathbf{s}, i)][\text{sgn } z_-(\mathbf{s}, i)] \leq 0 \\ 0 & : \text{otherwise,} \end{cases} \quad (24)$$

in which we have defined $\text{sgn } 0 = 0$. If $Q(\mathbf{s}, i) = 0$, then the point \mathbf{s} represents a local minimum in the direction of the i th dimension, and so that dimension should be deactivated. (Note that this definition is correct only because we know that $\gamma(\mathbf{s})$ has a *positive* step at zero, and hence cannot cause a local *maximum*.)

The Algorithm Itself First, inactivate all dimensions by setting

$$I_0 = \{1, \dots, M\}, \quad I_1 = \emptyset, \quad \mathbf{s} = \mathbf{0}.$$

Then, for each iteration of the main loop:

1. Compute proposed new point \mathbf{s}' and step $\mathbf{\Delta}$ according to quasi-Newton or conjugate gradient algorithm, so that $\mathbf{s}' = \mathbf{s} + \mathbf{\Delta}$.
2. See if the proposed step would result in any of the components of \mathbf{s} changing sign, by finding the set of all such zero-crossings:

$$Z = \{i \in I_1 : \text{sgn } s_i \neq \text{sgn } s'_i\}$$

3. See if any of the zero-crossings satisfy the inactivation criterion. First we define $\lambda(i)$ as the step size that will take us to the zero crossing in the i th dimension:

$$\lambda(i) = -s_i/\Delta_i,$$

so that $[\mathbf{s} + \lambda(i)\mathbf{\Delta}]_i = 0$. Then we find the set Z_0 :

$$Z_0 = \{i \in Z : Q(\mathbf{s} + \lambda(i)\mathbf{\Delta}, i) = 0\}.$$

4. If there are any such zero crossings, choose one (*e.g.* the last along the line from \mathbf{s} to \mathbf{s}') and truncate the step there, otherwise, take the proposed step unmodified. (We are using \leftarrow to denote assignment.)

$$\lambda^* = \begin{cases} \min_{i \in Z_0} \lambda(i) & : Z_0 \neq \emptyset \\ 1 & : Z_0 = \emptyset \end{cases}$$

$$\mathbf{s} \leftarrow \mathbf{s} + \lambda^* \mathbf{\Delta}.$$

5. Update the active and inactive sets to reflect the new current point \mathbf{s} . To do this, we define two sets:

$$I_- = \{i \in I_1 : Q(\mathbf{s}, i) = 0 \wedge s_i = 0\}$$

$$I_+ = \{i \in I_0 : Q(\mathbf{s}, i) = 1\}.$$

I_- is the set of currently active dimensions that should be deactivated. I_+ is the set of currently inactive dimensions eligible for reactivation. We pick just *one* of these. It is not clear how to choose the one to be released, as the optimiser may end up in one of several local minima depending on the choice, but we attempt to rate their ‘eligibility’, reassigning I_+ to contain only the dimension with the highest rating:

$$I_+ \leftarrow \{\arg \max_{i \in I_+} \frac{1}{2} |z_+(\mathbf{s}, i) + z_-(\mathbf{s}, i)|\}.$$

All that remains is to transfer the various elements between the active and inactive sets:

$$I_0 \leftarrow I_0 \cup I_- \setminus I_+$$

$$I_1 \leftarrow I_1 \cup I_+ \setminus I_-,$$

where the operator \setminus denotes set subtraction.

6. Perform any book-keeping required by the quasi-Newton algorithm—in particular update the Hessian approximation using gradient information at the current point. The important point about this is that we update only the sub-matrix defined by the currently active dimensions.



Figure 6: Individual patterns of the modified bars data set

The main loop is subject to the same termination conditions as the unmodified optimiser, except that these apply to the currently active dimensions only. In addition, we terminate if there are no active dimensions.

By inactivating directions in which the local gradient is discontinuous, this algorithm keeps the Hessian approximation from becoming ill-conditioned. Under certain conditions, this algorithm results in significant speed improvements over the equivalent unconstrained quasi-Newton optimiser (see the results of the next section) essentially by exploiting the sparsity we are trying to achieve. Indeed, in use, as the basis converges, the code becomes sparser and the performance improves.

In addition, the quasi-Newton version of this algorithm performed better than the equivalent active set conjugate gradient version. These two were, as far as possible, evenly matched, using the same termination conditions and the same line search procedure. It seems that the potential expense of maintaining a large inverse Hessian approximation is mitigated by the fact that only a small number of dimensions is active at any one time.

7 The Bars Problem

We tested the algorithm on a variation of the bars data set [Földiák, 1990], using 16-dimensional pattern vectors arranged into 4 by 4 shapes (see figure 7). Note that even though there are 13 out of a possible 16 linearly independent patterns, they still form an intrinsically overcomplete set in the sense that they span only a 9-dimensional subspace. In other words, the patterns are linearly dependent, as the following ‘equations’ graphically demonstrate:

$$\begin{array}{c}
 \begin{array}{ccccccc}
 \begin{array}{|c|} \hline \blacksquare \\ \hline \end{array} & + & \begin{array}{|c|} \hline \blacksquare \\ \hline \end{array} & + & \begin{array}{|c|} \hline \blacksquare \\ \hline \end{array} & + & \begin{array}{|c|} \hline \blacksquare \\ \hline \end{array} & = & 2 \begin{array}{|c|} \hline \blacksquare \\ \hline \end{array} & + & \begin{array}{|c|} \hline \blacksquare \\ \hline \end{array} \\
 \end{array} \\
 \\
 \begin{array}{ccccccc}
 \begin{array}{|c|} \hline \blacksquare \\ \hline \end{array} & + & \begin{array}{|c|} \hline \blacksquare \\ \hline \end{array} & = & \begin{array}{|c|} \hline \blacksquare \\ \hline \end{array} & + & \begin{array}{|c|} \hline \blacksquare \\ \hline \end{array} \\
 \end{array}
 \end{array}$$

The actual input patterns were linear combinations of these patterns, the mixing coefficients being drawn from a sparsified Laplacian distribution, that is, a random mixture of Laplacians and zeros in the proportion $z : (1 - z)$. On top of this was added uniform Gaussian noise of variance σ_* . The parameters z and σ_* were varied between experiments.

Both a Laplacian prior and the sparsified Laplacian prior from Section 4 were tested using the modified quasi-Newton optimiser described in Section 6. The sparsified Laplacian prior was parameterised, as described in equation 20, in terms of μ and K , various combinations of which were tested. The results were compared with those obtained using a family of smooth sparse priors:

$$p(s) \propto \text{sech}^{1/\beta} \beta s, \quad (25)$$

$$\gamma(s) = \tanh \beta s. \quad (26)$$

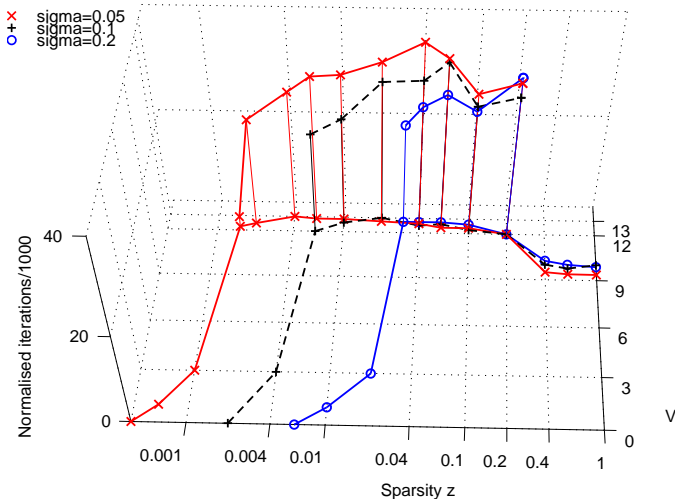


Figure 7: Average numbers (V) of non-zero basis vectors learned using modified optimiser with Laplacian prior, for several sparsities, z , and three values of σ . The vertical axis shows the average number of iterations, normalised to an effective learning rate of 0.016, taken to converge to the correct 13 basis vectors.

As the ‘sharpness’ parameter $\beta \rightarrow \infty$, this tends to a Laplacian distribution, but for finite β , the distribution and $\gamma(s)$ both remain smooth, so that a standard quasi-Newton optimiser is sufficient to find the maximum of the posterior.

The matrix Λ_ϵ was fixed at $\sigma^{-2}\mathbf{I}$, varying σ between experiments. Learning was performed using batches of 96 input patterns chosen randomly each time (*i.e.* learning was not performed repeatedly on the same batch.) The square (16 by 16) basis matrix A was initialised to $0.1\mathbf{I}$ (rather than \mathbf{I}) at the start of each run, in order to avoid an initial period during which the weights simply decayed to roughly the right magnitudes before finding the right directions. The learning rate η was of the order of 0.01, but it was found that smaller learning rates were required for high sparsities (*i.e.* small z).

A number of observations can be made about the results:

Effect of low sparsity Though there were 13 distinct patterns, the basis did not always converge to 13 vectors. In the case of non-sparsified Laplacian input (*i.e.* $z = 1$) it nearly always converged to 9 or 10 basis vectors, which is interesting because the intrinsic dimensionality of the data set is 9. It seems that a certain level of sparsity is required for the algorithm to find an overcomplete basis.

Effect of high sparsity Conversely, too high a level of sparsity also resulted in too few basis vectors being found—the scaling of these vectors is roughly proportional to z , the proportion of Laplacians in the source mixture density. As the sparsity is increased, at a certain point, all the basis vectors collapse to zero. The point at which this happens depends on the noise parameter σ —the higher the value of σ , the higher the minimum sparsity.

Performance of modified optimiser The performance advantage of the modified quasi-Newton optimiser with zero-crossing detection kicks in only

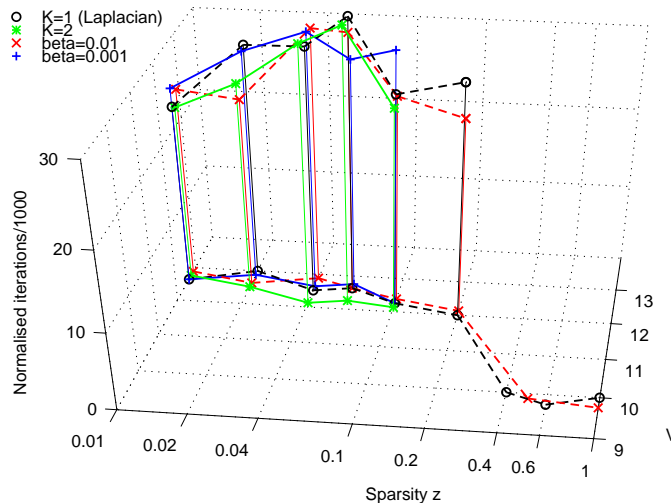


Figure 8: Comparison of learning behaviour using standard and modified optimisers for two values of β (see equation 25) and two values of K . Setting $K = 1$ corresponds to an unmodified Laplacian prior.

when significant numbers of units can be set to exactly zero. The conditions under which this happens involve a trade-off between the sparsity of the input, the actual noise level σ_* , the expected noise level σ , and the form of the prior.

The standard quasi-Newton optimiser using the parametrised smooth prior (equation 25) is fast for large β , but slows down as β decreases and the prior becomes more sharply peaked (results not shown due to lack of space). Holding the other parameters fixed, there will be a value of β_c such that if $\beta < \beta_c$, the modified optimiser will out-perform the standard one.

Effect of modified prior The sparsified prior has an effect in ‘cleaning-up’ the output, encouraging more of the outputs to remain at exactly zero, reflecting more accurately the actual source distribution—a mixture of Laplacians and zeros. It also results in much faster performance from the modified optimiser.

However, enforcing this greater degree of sparsity too aggressively seems to disrupt learning, resulting in too few basis vectors being learned. This is probably because the shape of the prior admits of a multimodal posterior, with strong local maxima at coordinate zeros. Units that should be activated fail to come on, which stops them from participating in the learning process.

The overall conclusion is that the algorithm is capable of discovering a sparse code to represent the input that accurately reflects the unknown number of independent basic patterns. The patterns themselves may be linearly dependent and hence form an overcomplete basis within the subspace spanned by them. The success of this is, however, dependent on the sparsity of the input being between certain limits—too low, and an overcomplete basis may not be found; too high, and the basis vectors decay to zero.

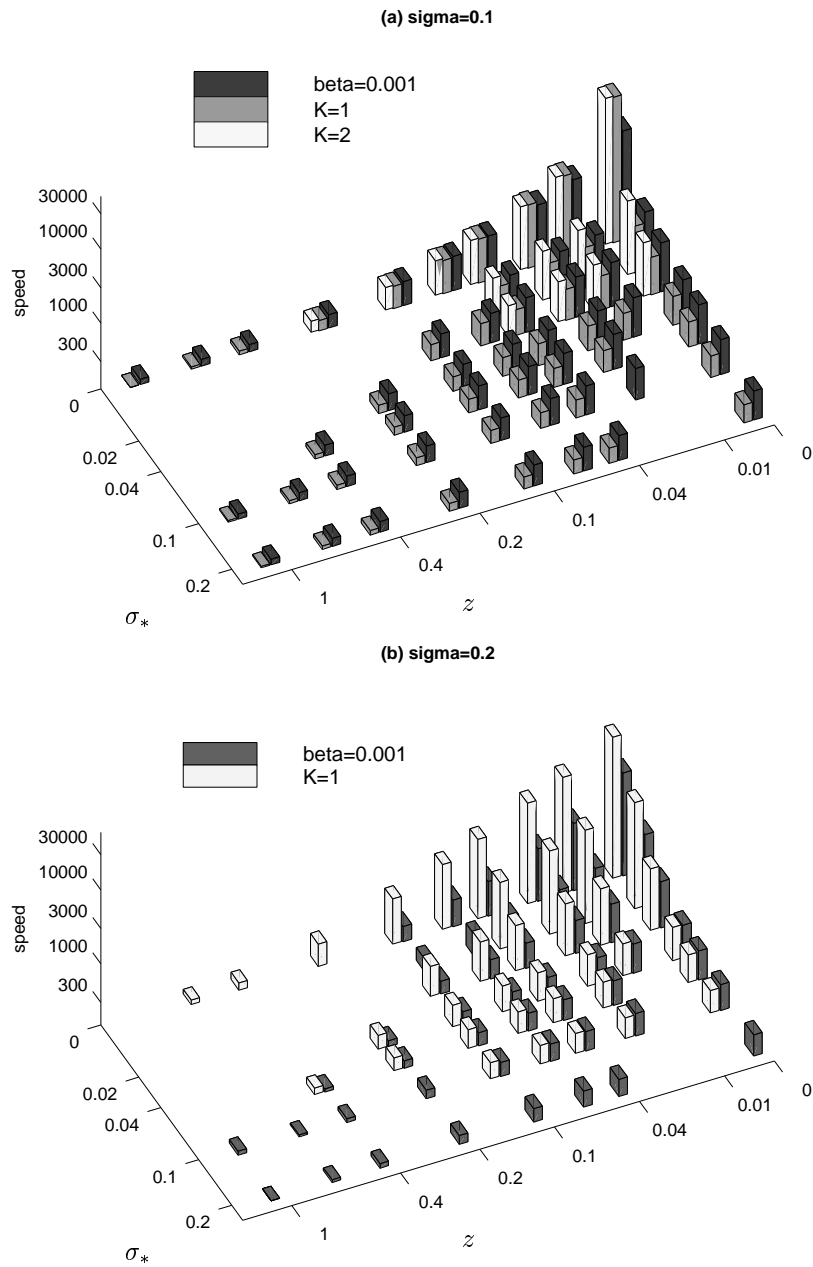


Figure 9: Performance of standard and modified optimisers under various conditions. These results were obtained by running the optimiser with the basis matrix \mathbf{A} equal to the actual basis used to generate the data. The speeds are quoted in optimisations per second. The upper chart (a) shows how the modified optimiser is faster for high sparsities and low input noise levels, while lower chart (b) illustrates that this effect is more pronounced at high values of σ , the expected noise parameter. In this particular (rather small) problem, the modified optimiser outperforms the standard one over only a restricted range of parameters and is of questionable usefulness. Further experiments with larger problems have shown it in a more favourable light.

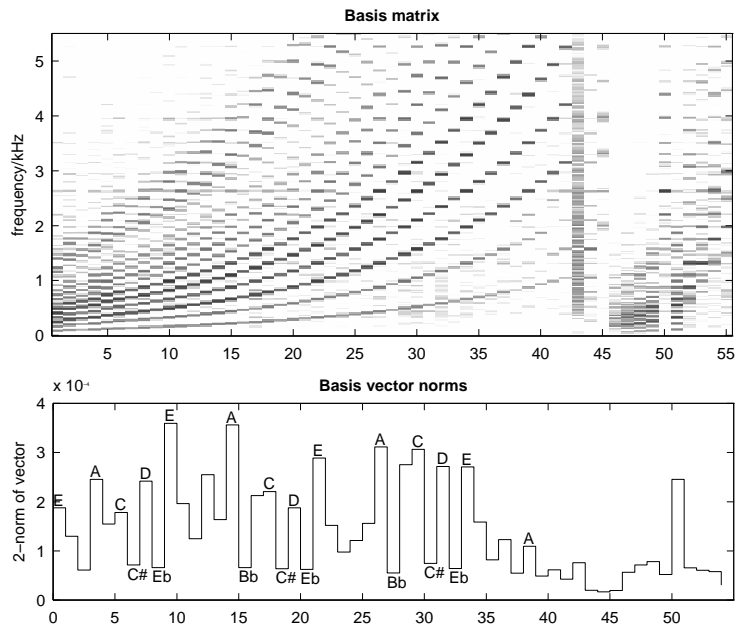


Figure 10: Basis vectors and their norms

8 Application to Synthesised Music

We tested the algorithm on real sounds generated by a MIDI synthesiser and then resampled. We used a MIDI synthesised piece of music, Bach’s Partita in A minor for keyboard, BWV827, chosen because it consists mainly of two or three relatively independent lines with few block chords. The hope was that the ‘independent components’ would turn out to be the notes themselves.

We used magnitude spectra rather than power spectra in order that the noise be more approximately Gaussian and to reduce the super-Gaussianity of the marginals of spectral bands, as illustrated in figure 3. We used our modified quasi-Newton optimiser with the sparsified Laplacian approximation described in Section 4. We tried both Harpur’s Hebbian learning rule and Lewicki and Sejnowski’s covariant algorithm, but found that the former was more stable in the early stages of learning. We then switched to the covariant rule after the basis vectors had settled down.

It learned 55 basic spectra, of which 49 were note spectra. The pattern of harmonics in each basis vector appeared to match the notes in the piece. In order to find out what they sounded like, we developed a technique for ‘re-animating’ the spectra. For each basis vector, we built an FIR digital filter with the equivalent frequency response and then drove it with white noise. This was not intended to be a realistic reconstruction of the input sounds—only to give us an idea of what was going in the system. Generally speaking, there is no requirement for a perceptual system to be able to make physically accurate reconstructions of its input, and this is not the primary goal of our work.

Realism aside, most of the basis vectors produced clearly pitched sounds¹,

¹A selection of these sounds and other related material, including some

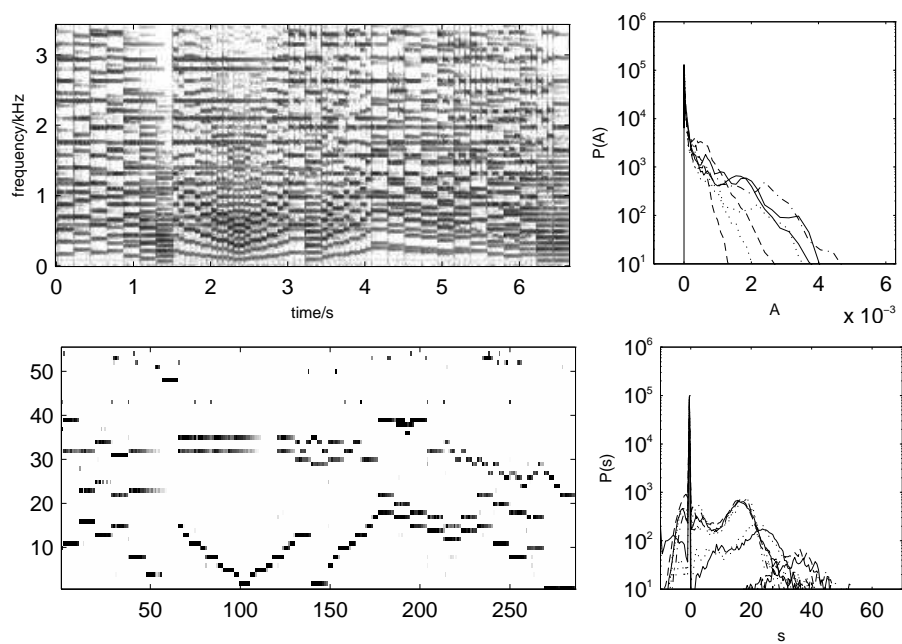


Figure 11: Top: Input spectrogram and histograms of activity at a selection of frequencies. Bottom: output and output histograms. The output is visibly sparser than the input. The output histograms also show some interesting structure: their bimodality reflects the on/off nature of the notes present in the music.

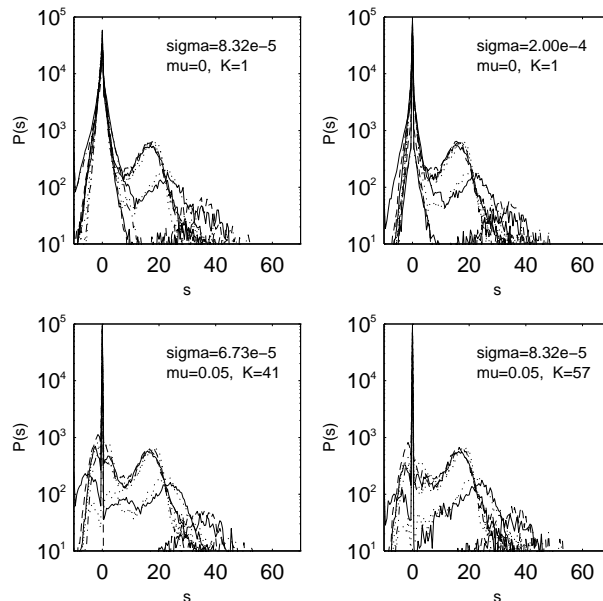


Figure 12: Output marginals for various parameter settings. Each line shows the distribution of activity of a single output unit. See text for discussion.

all of which matched notes in the music. A few of the basis vectors produced indistinct sounds at the bottom of the pitch range, where the resolution of spectrogram was insufficient to resolve the harmonics reliably. The corresponding output units did, however, reliably respond to the lower notes in the music. One of other the vectors corresponded to wide-band noise, and responded mainly to note onsets.

On the basis of these sounds, we were able manually to label and re-arrange the basis vectors into note-order to produce figure 7. The zero-length vectors were discarded, leaving 55 vectors. The relative strengths of the basis vectors were related to the corresponding note probabilities, with more common notes represented by larger vectors. Interestingly enough, the key of the piece (A minor) was reflected in the profile of basis vector norms, with the most common notes (A and E) being more strongly learned than least ($B\flat$, $C\sharp$, $E\flat$).

Once trained, the outputs certainly looked like a moderately faithful ‘piano-roll’ transcription of the music (see figure 8). To check, we used them to drive the MIDI synthesiser, triggering notes on zero-to-nonzero transitions. After some trial and error adjustments of the prior parameters μ and K , this resulted in a passable rendition² of the original piece, perhaps by a rhythmically-challenged piano student!

Whilst encouraging, this did point out the sensitivity of the procedure to the parameters of the prior. A large K tended to produce a clean output, but with many missing notes. A small K recovered the missing notes, but also allowed many small activations to come through, triggering notes that weren’t in the demonstrations of the system implemented in Java may be found at <http://www.eee.kcl.ac.uk/~samer/research/sparsecoding/>.

²Available at <http://www.eee.kcl.ac.uk/~samer/research/sparsecoding/bach-rec.mid>.

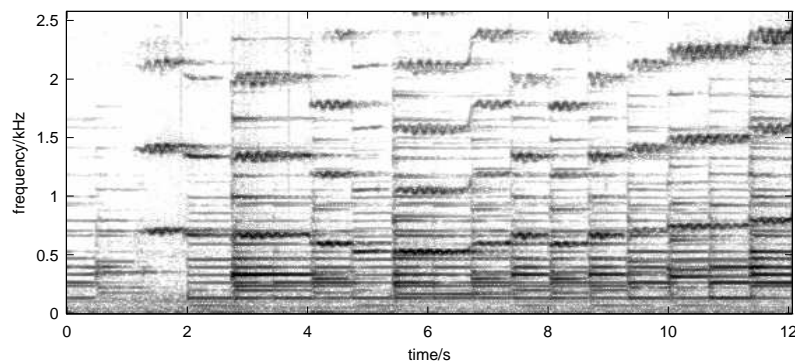


Figure 13: A spectrogram of the opening of Beethoven's *Kreutzer Sonata* for violin and piano. The two instruments make very distinctive patterns when viewed in two-dimensions. For example, the violin displays a lot of vibrato and the occasional slide between notes (*e.g.* at about 6.7s in).

music. Imposing a threshold of activity for triggering notes improved matters, but seemed a little inelegant, especially after the discussion of Section 4 about the desire for exact zeros to signal inactivity.

Histograms of output activity for different parameter settings (see figure 9) illustrate, amongst other things, how enforcing greater sparsity suppresses low level activity. They also show how the different normalisations of the basis vectors produce essentially similar output distributions but on different scales. In addition, many of the distributions appear distinctly bimodal, suggesting (accurately in this case) some underlying binary process. The fact that the distributions have differing widths shows that the choice of a single activation threshold for all the output units is not ideal. A better approach would be to use an adaptive threshold on each unit, which would find an optimal triggering point somewhere in the valley between the two modes.

9 Discussion

We have seen that a very simple sparse coder, working on instantaneous spectra, is capable of polyphonic note detection. There are a number of ways in which the system could be improved.

Adaptive thresholding of outputs An obvious development would be to use the marginal distributions of the outputs to choose a suitable threshold for note activation. This could be done by maximising the mutual information between the real-valued activity and the binary output obtained after thresholding, perhaps with the assumption that the real value is noisy [Orwell and Plumbley, 1999].

Learning about temporal variations The use of instantaneous spectra (1D patterns) is reasonable for sounds with constant timbres, but even so, there are still problems because there is no concept of the temporal coherence of a note. This is apparent in the way that notes are sometimes broken up over time or have ragged onsets.

Notes of different instruments have distinctive shapes in the time-frequency plane (see figure 10 for an example). An algorithm trained on *two*-dimensional spectrogram patches may be able to learn these shapes, and should result in more reliable note detection and a much better ability to discriminate different instruments.

The main drawback with this is that the input vectors immediately become much larger: for example, if the instantaneous spectra have 256 components, and we choose to train the coder on strips that are 32 points wide, then the input patterns will have 8192 elements! A possible short cut would be to restrict ourselves to *separable 2D patterns*. Imagine a 2D shape in time and frequency: $\psi(\omega, t)$. If this is expressible as a product, $\psi(\omega, t) = \phi(\omega)\chi(t)$, then we should be able to learn these patterns one dimension at a time. We would first code patterns in frequency as we do now, then we would code the temporal patterns of each output independently. We could either train the same coder on all outputs, in which case it would develop a dictionary of activity envelopes applicable to all notes, or we could train a separate coder on each output, so that each one would be specifically adapted to the note on which it was trained. For example, if high notes tended to be short and percussive, but low notes tended to be long and sustained, then the coders trained on high and low notes would learn different bases, composed predominantly of short and long envelopes respectively. We expect that this would go a long way towards improving the temporal coherence of notes with relatively constant spectral profiles, though it will probably be unable to deal with vibrato.

An alternative way of dealing with timbral variation is to consider the *temporal evolution subspace* of a note. Imagine the instantaneous spectrum of a sound as a point in the high dimensional input space: As the sound begins, evolves and ends, this point traces a path starting and finishing at the origin (see figure 11). This is similar to the *timbre tracks* introduced by Godsmark and Brown [Godsmark and Brown, 1999], except that rather than confine the track to a *fixed* two-dimensional subspace defined by amplitude and spectral brightness, we would allow the track to wander through the many (unnamed) dimensions of the original input space. We could then fit a lower-dimensional subspace to the path, with different combinations of pitch and instrument creating paths that fit into different subspaces. The subspace defined by amplitude and brightness is just one of the possible subspaces; timbre tracks represent the

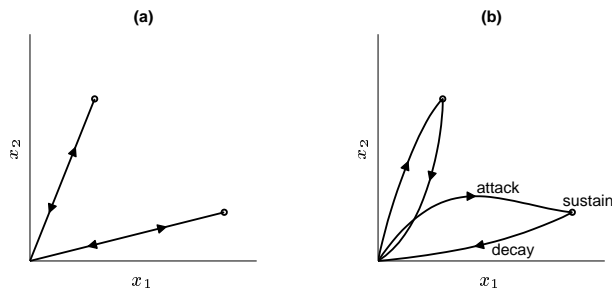


Figure 14: Temporal evolution of the (hypothetical) instantaneous spectra of two different sounds with (a) constant spectral envelopes, fitting into 1D subspaces; (b) more complex sounds, fitting into 2D subspaces.

projection of the full path onto that subspace.

The current system can deal with paths that fit into a one-dimensional subspace; that is, a line. (The direction of this line is represented by the direction of the appropriate basis vector.) However, it may be possible to combine elements of Hyvärinen’s Subspace ICA algorithm [Hyvärinen, a] with the current sparse coder, so that it can deal with higher dimensional subspaces. The output units would then represent activity within a learned subspace.

A better noise model The sparse coder we have implemented assumes additive spherical Gaussian noise on the input ($\mathbf{A}_\epsilon = \sigma^{-2}\mathbf{I}$), which is not very accurate for audio spectrograms. Even if the original signal is noiseless, the spectrogram shows a background of noise-like activity, due to *spectral leakage*, an artefact of the windowed Fourier transform applied to aperiodic signals. The activity at a particular frequency is related to the energy in the signal in the region of that frequency. Since there tends to be more energy at the low end, the ‘noise’ is more pronounced at low frequencies, and slopes off towards the high. This means that if the noise level σ is set high enough to prevent the system being distracted by high level noise at low frequencies, it fails to pick up significant low level details at high frequencies, where the actual noise level is lower. One option is to set manually some noise distribution *a priori*. Another is to estimate the noise covariance matrix from the data.

Achieving greater sparsity An analysis of the learning behaviour of this system shows that the sparsified Laplacian approximation does not actually work as intended (see [Abdallah and Plumbley, 2000]). Though the piece-wise Laplacian density appears visually to be a reasonable approximation to a sparsified Laplacian (that is, a Laplacian plus a delta distribution), it is actually a poor one for this application.

A better way to deal with greater sparsity is to work directly with a mixed discrete/continuous prior, recognising that zero and non-zero activities form two distinct *discrete* states. Olshausen and Millman [Olshausen and Millman, 1999] do this by modelling the prior as a parametrised mixture of Gaussians, introducing M new discrete state variables. If, for example, we chose to model each source as a mixture of two Gaussians, we would introduce M binary variables u_i . The prior for each source would be given by

$$P(s_i) = \sum_{u_i \in \{0,1\}} P(s_i|u_i)P(u_i),$$

where $P(u_i)$ represents a binary distribution, and $P(s_i|u_i=0)$ and $P(s_i|u_i=1)$ represent the two independently parametrised Gaussian distributions that go into the mixture. (This is not the notation used by Olshausen and Millman.)

The joint probability of all the variables s_i and u_i would be given by

$$P(\mathbf{s}, \mathbf{u}) = \prod_{i=1}^M P(s_i|u_i)P(u_i).$$

On substituting this into equation 2, we would find that the overall data likelihood $P(x|\theta)$ (where θ now represents *all* the parameters, including \mathbf{A}) breaks into a sum of Gaussian integrals. This renders the integrals tractable, but still

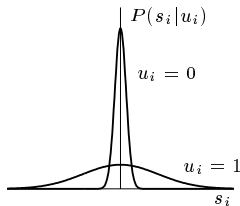


Figure 15: Mixture-of-Gaussians prior, made up of two Gaussians with binary state variables u_i . (After figure 2 of [Olshausen and Millman, 1999].)

leaves the problem of summing over an exponential number of discrete states. Lewicki and Sejnowski’s solution is to use Gibbs sampling to visit discrete states that cover most of the significant volume of the posterior. Being a stochastic method, this requires that, for each presentation of an input pattern, several Gibbs iterations are required to collect statistics about the posterior.

During inference, instead of finding the maximum of the posterior distribution over \mathbf{s} , that is, the maximum of $P(\mathbf{s}|\theta, \mathbf{x})$, they find the most likely values of the binary state variables first, and then estimate \mathbf{s} given the estimate of \mathbf{u} :

$$\hat{\mathbf{u}} = \arg \max_{\mathbf{u}} P(\mathbf{u}|\mathbf{x}, \theta)$$

$$\hat{\mathbf{s}} = \arg \max_{\mathbf{s}} P(\mathbf{s}|\mathbf{x}, \mathbf{u}, \theta).$$

If we were to replace the first Gaussian (for $u_i = 0$) in each mixture with a delta distribution at zero, so that $P(s_i|u_i=0) = \delta(s_i)$, the procedure would be essentially unchanged, but we would get the mixed discrete/continuous sparse prior we were after.

This procedure should confer a number of benefits, including improved basis vector normalisation, a cleaner and more faithful output, reduced numbers of basis vectors going to zero, and an ability to learn highly overcomplete representations.

10 Conclusions

The main conclusion of this work is that there *is* enough structure in music (or at least certain kinds of music) for a sparse coder learn about and detect notes in an unsupervised way, even when the music is polyphonic. There is no need to bring any prior musical knowledge to the problem, such as the fact that musical notes have approximately harmonic spectra. This knowledge is acquired during learning. In addition, the MAP activation framework using a sparse prior does a reasonably good job of decomposing an input spectrum into a sum of notes.

The sparse coding model we used is essentially an extension of ICA, so one might conclude that “notes are the independent components of music.” Clearly, the notes of a piece of music are *not* independent, but those dependencies are not captured in the linear generative model we have been considering here.

One of more the interesting conclusions to be drawn from the experiment with the bars problem is that a certain amount of sparse structure is required in the input if the system is to learn a highly overcomplete basis. If this is not the case, the system learns only enough basis vectors to span the input space.

On a less encouraging note, the approximate sparsified Laplacian prior did not seem to have the desired effect. The main problem is that the algorithm, being based on a unimodal approximation to the posterior, performs less reliably for very sparse priors where the posterior is multimodal. Not only does it get stuck in local minima (resulting in missing notes) but the approximations used in the derivation of the learning rule break down. This may account for the instability of the Lewicki and Sejnowski learning rule equation 19 under certain conditions.

In addition, the mismatch between the prior and the actual source distributions leads to poor scaling of the basis vectors, in some cases decaying to zero for very rarely active input patterns.

Overall, these results lend support to the notion that the principle of redundancy reduction, initially via the method of sparse coding, can reproduce and explain some of the features of human auditory perception.

A Appendix on Spectrograms

A.1 Computational Details

Let the audio signal be $\psi(t)$, and the sampling period be T . We take successive overlapping windows of $2L$ samples, (so that the spectrogram will have L frequencies) moving the window H samples at a time. The p th window of samples, arranged into a vector \mathbf{u} , is given by

$$u_j(p) = \psi(T[Hp + j]), \quad 1 \leq j \leq 2L. \quad (27)$$

The vector \mathbf{h} contains the Fourier analysis window. We use a Hamming window, defined as

$$h_j = 0.54 - 0.46 \cos \frac{\pi j}{L}.$$

The complex DFT coefficient v_k for the k th frequency is given by

$$v_k = \sum_{j=1}^{2L} \left[\exp \frac{i\pi}{L} jk \right] h_j u_j, \quad (28)$$

where $i = \sqrt{-1}$, and we have dropped the index p for clarity. The power in the k th frequency is obtained by taking the magnitude squared of the DFT coefficient, but we actually use the magnitude itself as the input \mathbf{x} to our sparse coder:

$$x_k = |v_k| = \sqrt{v_k^* v_k}, \quad (29)$$

where the $*$ operator denotes complex conjugation. The k th frequency itself is $\omega_k = k/2LT$.

A.2 Noise Statistics

Consider the DFT of equation 28 as a matrix operation on a window of samples. If the analysis window is rectangular (*i.e.* $\forall j, h_j = 1$), then the relevant matrix is orthogonal (up to a scaling), meaning that spherical Gaussian noise at the input appears as spherical Gaussian noise at the output. There will be two

components per frequency ω ; let us call them X_ω and Y_ω . (These will be represented in the real and imaginary parts of the complex DFT.) The energy in the spectrum at that frequency can be expressed as

$$S_\omega = X_\omega^2 + Y_\omega^2.$$

If the input consists purely of noise, then these two components will be Gaussian and uncorrelated; hence their 2-dimensional joint distribution will be Gaussian and circular. Now, for two identically distributed Gaussian random variables X and Y , both with variance $1/\lambda$, one can show that $R = \sqrt{X^2 + Y^2}$ is another random variable distributed according to

$$p(r) = \lambda r e^{-\frac{1}{2}\lambda r^2}. \quad (30)$$

If we then let $S = R^2 = X^2 + Y^2$, we find that

$$p(s) = p(r) \frac{dr}{ds} = \frac{1}{2} \lambda e^{-\frac{1}{2}\lambda s}, \quad (31)$$

which is a (single-sided) exponential distribution as stated in Section 2. This does show, however, that if we were to use instead the *magnitude* spectrogram (that is, the square-root of the power spectrogram) we would get a noise distribution equivalent to equation 30, which has, at least, an approximately Gaussian tail.

Note that this analysis does not apply to non-rectangular analysis windows. It is only intended as a rough heuristic argument for using the magnitude spectrum rather than the power spectrum.

References

- [Abdallah and Plumbley, 1999] Abdallah, S. A. and Plumbley, M. D. (1999). Unsupervised learning for music perception. In *Cambridge Music Processing Colloquium*, Cambridge, England. Online at <http://www.eee.kcl.ac.uk/~samer/docs/cmc99.ps.zip>.
- [Abdallah and Plumbley, 2000] Abdallah, S. A. and Plumbley, M. D. (2000). Analysis of a one dimensional ‘sparse coder’. In preparation.
- [Atick, 1992] Atick, J. J. (1992). Could information theory provide an ecological theory of sensory processing? *Network: Computation in Neural Systems*, 3(2):213–251.
- [Barlow, 1989] Barlow, H. B. (1989). Unsupervised learning. *Neural Computation*, 1:295–311.
- [Cardoso, 1997] Cardoso, J.-F. (1997). Information-Maximisation and Maximum Likelihood for Blind Source Separation. *IEEE Signal Processing Letters*, 4(4):112–114.
- [Duda et al., 1990] Duda, R., Lyon, R., and Slaney, M. (1990). Correlograms and the separation of sounds. In *Proc. IEEE Asilomar Conf. on Signals, Systems and Computers*.
- [Endres and Földiák, 1999] Endres, D. and Földiák, P. (1999). Quadratic programming for learning sparse codes. In *Intl. Conf. on Artificial Neural Networks*, pages 593–596, Edinburgh.
- [Field, 1994] Field, D. J. (1994). What is the goal of sensory coding? *Neural Computation*, 6:559–601.

- [Field and Olshausen, 1996] Field, D. J. and Olshausen, B. A. (1996). Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381:607–609.
- [Flandrin and Rioul, 1990] Flandrin, P. and Rioul, O. (1990). Affine smoothing of the wigner distribution. In *ICASSP'90*, pages 2455–2458.
- [Földiák, 1990] Földiák, P. (1990). Forming sparse representations by local anti-Hebbian learning. *Biological Cybernetics*, 64(2):165–170.
- [Gabor, 1947] Gabor, D. (1947). Acoustical quanta and the theory of hearing. *Nature*, 159:591–594.
- [Godsmark and Brown, 1999] Godsmark, D. and Brown, G. J. (1999). A blackboard architecture for computational auditory scene analysis. *Speech Communication*, 27:351–366.
- [Harpur, 1997] Harpur, G. F. (1997). *Low Entropy Coding with Unsupervised Neural Networks*. PhD thesis, Cambridge University Engineering Department.
- [Hyvärinen, a] Hyvärinen, A. Emergence of phase and shift invariant features by decomposition of natural images into independent feature subspaces. *Neural Computation*, 12(7):1705.
- [Hyvärinen, b] Hyvärinen, A. Survey on Independent Component Analysis. *Neural Computing Surveys*, 2:94.
- [Hyvärinen, 1998] Hyvärinen, A. (1998). Sparse code shrinkage: Denoising of nongaussian data by maximum-likelihood estimation. Technical report, Helsinki University of Technology.
- [ichi Amari, 1998] ichi Amari, S. (1998). Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276.
- [Jordanov and Penev, 1999] Jordanov, L. G. and Penev, P. S. (1999). The principal component structure of natural sound. In *Advances in Neural Information Processing Systems*, volume 11. M.I.T. Press.
- [Karhunen,] Karhunen, J. Neural approaches to Independent Component Analysis and source Separation.
- [Lewicki and Sejnowski, 1998] Lewicki, M. S. and Sejnowski, T. J. (1998). Learning overcomplete representations. *Neural Computation*.
- [MacKay, 1996] MacKay, D. J. C. (1996). Maximum likelihood and covariant algorithms for independent component analysis.
- [Olshausen, 1996] Olshausen, B. A. (1996). Learning linear, sparse, factorial codes. Technical Report A.I. Memo 1580, MIT.
- [Olshausen and Millman, 1999] Olshausen, B. A. and Millman, K. J. (1999). Learning sparse codes with a mixture-of-gaussians prior. In *Advances in Neural Information Processing Systems*, volume 11. M.I.T. Press.
- [Orwell and Plumbley, 1999] Orwell, J. and Plumbley, M. D. (1999). Maximizing information about a noisy signal with a single non-linear neuron. In *Intl. Conf. on Artificial Neural Networks*, pages 581–586, Edinburgh.
- [Plomp, 1976] Plomp, R. (1976). *Aspects of Tone Sensation*. Academic Press.