



# Continuous speech recognition with sparse coding

W.J. Smit <sup>a,\*</sup>, E. Barnard <sup>b</sup>

<sup>a</sup> *University of Pretoria, Pretoria 0001, South Africa*

<sup>b</sup> *Meraka Institute, P.O. Box 395, Pretoria 0001, South Africa*

Received 30 July 2007; received in revised form 9 June 2008; accepted 15 June 2008

---

## Abstract

Sparse coding is an efficient way of coding information. In a sparse code most of the code elements are zero; very few are active. Sparse codes are intended to correspond to the spike trains with which biological neurons communicate. In this article, we show how sparse codes can be used to do continuous speech recognition. We use the TIDIGITS dataset to illustrate the process. First a waveform is transformed into a spectrogram, and a sparse code for the spectrogram is found by means of a linear generative model. The spike train is classified by making use of a spike train model and dynamic programming. It is computationally expensive to find a sparse code. We use an iterative subset selection algorithm with quadratic programming for this process. This algorithm finds a sparse code in reasonable time if the input is limited to a fairly coarse spectral resolution. At this resolution, our system achieves a word error rate of 19%, whereas a system based on Hidden Markov Models achieves a word error rate of 15% at the same resolution.

© 2008 Elsevier Ltd. All rights reserved.

*Keywords:* Sparse coding; Spike train; Speech recognition; Linear generative model

---

## 1. Introduction

The brain needs to form an internal representation of the outside world in order to interact with the world, and does so by representing or coding information in the activities of neurons. A neuron can be viewed as a binary element; it is either silent or it fires a spike. The activity pattern of several neurons over time is called a spatio-temporal pattern or a spike train. The brain uses these patterns to code stimuli.

What are the properties of the neural code? In a binary temporal system such as the brain there are two extremes of representing data: at the one end there is *dense coding*, where many neurons are very active in coding a stimulus. For dense coding, a small number of neurons are sufficient to code a large set of stimuli. At the other extreme there is *local coding*, where very few neurons are active. The extreme where just one neuron is active for a stimulus (1-of- $N$  coding) would require a large number of neurons to represent all the different

---

\* Corresponding author. Tel.: +27 18 2994027; fax: +27 18 2991320.

*E-mail addresses:* [willie.smit@gmail.com](mailto:willie.smit@gmail.com) (W.J. Smit), [ebarnard@csir.co.za](mailto:ebarnard@csir.co.za) (E. Barnard).

stimuli. The brain adopts a compromise between dense coding and local coding (Földiák and Young, 1995; Vinje and Gallant, 2000) which is called *sparse coding*.

Several studies have used sparse coding to create a feature set for sound or speech recognition tasks. The studies follow the same general approach: first a sound signal is encoded into a spike train, then the recognition task is performed by decoding the spike train. Cho and Choi (2005) perform sound classification with spikes. They classify a sound as belonging to one of ten classes, which include male speech, foot steps and flute sounds. Näger et al. (2002) show that transitions between vowels can be classified by learning the delays between spikes. Kwon and Lee (2004) use *independent component analysis* (ICA) to extract features from speech in order to do phoneme recognition. ICA is an algorithm that provides a sparse representation of a signal. Some studies illustrate isolated digit recognition (Loiselle et al., 2005; Mercier and Séguier, 2002; Verstraeten et al., 2005) while Holmberg et al. (2005) demonstrate isolated letter recognition. All of these studies consider only isolated samples.

In this article, we show how continuous speech recognition can be performed with sparse coding. We use a *temporal linear generative model* (TLGM) (Olshausen, 2002; Smith and Lewicki, 2005) to encode a speech signal into a spike train, although other encoding techniques could also be used. The TLGM is an extension of the *linear generative model* that has successfully been used to explain neural phenomena in the visual cortex (Olshausen and Field, 1997) and in the auditory cortex (Lewicki, 2002). The spike train is decoded by using the spike train model proposed by Oram et al. (1999). This study is an initial investigation into how continuous speech recognition can be done with sparse coding, and also what type of problems one encounters in the process. We would like to keep the recognition process simple, while still achieving reasonable results.

The recognition process we propose has three computational steps. Firstly, the raw waveform is transformed into a more natural representation, such as a spectrogram. Secondly, the modified representation is transformed into a sparse code. Finally, words are recognized by finding predetermined patterns in the sparse code. The following sections will cover each of these computational steps in turn.

## 2. Transforming the raw waveform

We use the TIDIGITS (Leonard and Doddington, 1993) dataset of continuous spoken digits by male and female speakers. The utterances are of variable length, consisting of a variable number of random digits. There are 11 different spoken digits, one for each number from “one” to “nine”, a “zero” and an “oh”. We choose this dataset because it is a rather simple set: it has a limited vocabulary and simple language model (since the probability that a certain word follows any other word is approximately equal for all the words).

The label data supplied with TIDIGITS is limited to orthographic transcriptions. That is, the digit sequence of each utterance is given, but the start- and endpoints of each digit in the waveform are not specified. We estimated these points by doing forced alignment with the *HVite* tool that is part of the *Hidden Markov Model Toolkit* (HTK) (Cambridge University, 2006). During forced alignment, HVite aligns a transcription with a waveform. The automatic forced alignment is accurate as HTK is able to model the data well: it is able to recognize almost all words in the test set correctly (word error rate<sup>1</sup> = 3%).

It is time consuming to train sparse coding models. We therefore use a reduced dataset so that the system can be trained in reasonable time. (With the reduced set, the entire system is trained in a week on 15 Pentium 4 PCs working in parallel.) The training set suggested for the TIDIGITS dataset has 8623 utterances and contains 28,329 words. Our training set is made up of every second sample of the suggested dataset. The reduced set contains at least 2000 samples of every word, and is therefore sufficient for our initial investigation.

The simplest way to represent a raw speech signal is as a time waveform. However, time domain representations of speech are not as natural as frequency domain representations, as is evidenced by the fact that the auditory pathway uses a frequency domain representation. Even so, sparse coding of the time waveform of speech signals has shown some correspondence with physiological data (Lewicki, 2002). There are models that provide frequency domain representations that to some extent agree with physiological measurements along the auditory pathway (Hermansky and Morgan, 1994; Wang and Shamma, 1995); however, for this study we

<sup>1</sup> word error rate =  $\frac{\# \text{ deletions} + \# \text{ insertions} + \# \text{ replacements}}{\text{number of words}} \times 100\%$ .

choose to use a simpler spectrogram representation. The spectrogram is constructed by first dividing the raw signal into segments. The segments span 25.6 ms (512 points with a sampling frequency of 20 kHz). A new segment starts every 20 ms or every 400 points, which gives an overlap between adjacent segments of 5.6 ms. Each segment is then windowed with a Hanning window before the log magnitude of the fast Fourier transform for that segment is calculated. The magnitude is chosen because the human ear is insensitive to phase information; the log of the magnitude is used because our perception of sound intensity is often approximated by the logarithm although we actually perceive sound intensity on a cube-root scale (Hermansky, 1990).

Humans do not perceive the content of a signal on a linear frequency scale. We therefore use a Mel-spaced filter-bank to incorporate this non-linearity. Each filter is a triangular bandpass filter whose centre frequencies are linearly spaced on a Mel-scale. Current automatic speech recognition systems use up to 32 filters, but we use only eight filters for computational reasons. A small number of filters is preferred because too many filters put significant computational strain on the algorithms, and accurate temporal information is apparently more important than accurate spectral information for speech recognition (Allen, 1994; Kral, 2000; Moller, 1999; Shannon et al., 1995). The outputs of the filters (with centre frequencies as given in Fig. 1) are then scaled so that the variance of each filter output over the dataset is one.

We use a temporal linear generative model to find the sparse code of a spectrogram. This model works best if important speech features have large values in the representation and if silences have values equal to zero. From inspection it seems for this particular dataset and the spectrogram transform described above, that values below 15 dB do not carry important speech features. We therefore subtract, after filtering, 15 dB from the spectrogram and set any negative value equal to zero.

Fig. 1 gives the signal representation for the utterance “one one one three eight eight one”. The spectrogram representation  $\mathbf{x}$  is an  $N_c \times N_t$  matrix with  $N_c$  the number of channels or filters and  $N_t$  the number of segments. It serves as input to the TLGM.

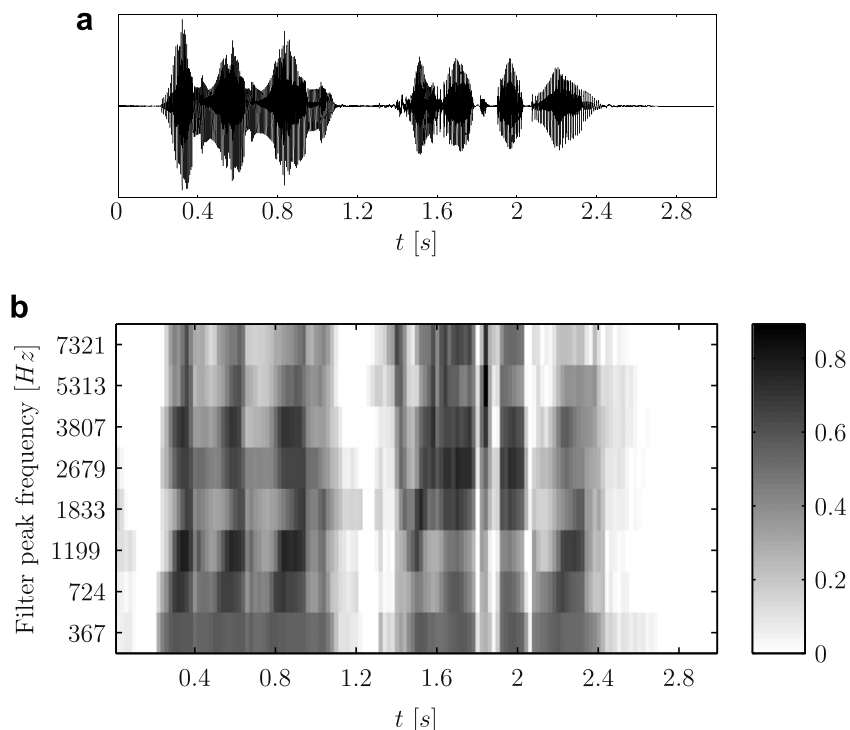


Fig. 1. (a) The speech waveform for the utterance “one one one three eight eight one”, and (b) its spectrogram representation. The gray scale indicates the signal amplitude (dB).

### 3. Sparse codes

We use a TLGM to encode a spectrogram into a spike train. The reconstructed signal (in our case a spectrogram)  $\hat{\mathbf{x}}$  is a convolution of the code  $\mathbf{a}$  with the dictionary  $\Phi$ . An element at time  $t$  in channel  $c$  of the reconstructed spectrogram is found with:

$$\hat{x}_{c,t} = \sum_{d=1}^{N_d} \sum_{\tau=1}^{N_t} \Phi_{c,\Delta t}^{(d)} a_{d,\tau} \quad (1)$$

For convenience, we use the integer index  $t$  to correspond to a segment in the spectrogram. The index  $t$  is related to actual time by multiplying  $t$  with 20 ms.  $N_t$  is the number of segments in the spectrogram that is being reconstructed.  $d$  is an index to a dictionary element; there are  $N_d = 224$  dictionary elements in our dictionary (we will explain why this number is chosen below).  $\Phi^{(d)}$  is an  $N_c \times \Delta\Phi$  matrix representing the  $d$ th dictionary element;  $\Phi_{c,\Delta t}^{(d)}$  refers to the element in  $\Phi^{(d)}$  located in channel  $c$  at time  $\Delta t$ .  $N_c$  is the number of channels in the signal, for the spectrogram used here  $N_c = 8$ .  $\tau$  is an index that refers to a segment (20 ms) of the code, i.e. a column of  $\mathbf{a}$ .  $\Delta t = \Delta\Phi - \tau + t$  where  $\Delta\Phi$  is the temporal width of a dictionary element. The width is taken as 260 ms (or 13 segments), as evidence points to auditory memory being around 250 ms in duration (Hermansky, 1998; Huggins, 1975; Massaro, 1972). When  $\tau$  is smaller than the temporal width of a dictionary element, the first  $\Delta\Phi - \tau$  columns of the dictionary element are truncated.  $\mathbf{a}$  is therefore an  $N_d \times N_t$  matrix. We will refer to  $a_{i,j}$  as a code element.

A non-zero code element corresponds to a spike in the neural code. Biological spikes signal binary events, whereas the TLGM yields real valued spikes. We can interpret positive real values as some number that is related to the number of neurons that fire spikes simultaneously. The brain appears to be redundant in this manner: more than one neuron can code the same information, which helps the brain to deal with physical damage. Negative real values are more difficult to interpret, although they can be viewed as inhibitory spikes. We choose to rather constrain the code elements to be non-negative  $a_i \geq 0$ . Later on we show that this constraint has an algorithmic advantage.

#### 3.1. Finding the sparse code

The code has to be a fair representation of the signal while it should also be sparse. A common approach (see for example Olshausen and Field (1997); Lewicki (2002)) is to use mathematical optimization to find the best code  $\mathbf{a}^*$

$$\mathbf{a}^* = \min_{\mathbf{a}} E(\mathbf{a}) \quad (2)$$

with

$$E(\mathbf{a}) = \|\mathbf{x} - \hat{\mathbf{x}}\|^2 + \lambda S(\mathbf{a}) \quad (3)$$

$\lambda$  is a parameter that sets the balance between the reconstruction error  $\|\mathbf{x} - \hat{\mathbf{x}}\|^2$  and the sparseness of the code  $S(\mathbf{a})$ .

The sparseness function should have a suitable form in order to encourage sparse codes (Kreutz-Delgado et al., 2003). We use the following sparseness function:

$$S(\mathbf{a}) = \sum_j \sum_i g(a_{i,j}) \quad (4)$$

In the case of non-negative codes, sufficient conditions to favour sparse codes are that  $g(a)$  be a concave function and monotonically non-decreasing. Here we use  $g(a) = a - \frac{\beta}{2}a^2$  with  $\beta = 0.1$ , which is a valid sparseness function for values of  $a$  less than  $1/\beta$  (it is monotonically decreasing beyond  $1/\beta$ ). The sparseness function in Eq. (4) makes the minimization problem in Eq. (2) quadratic. It can now be solved efficiently with a suitable bound constrained quadratic programming algorithm. The bounds are  $a \geq 0$  (which fits a neural code) and  $a \leq 1/\beta$ . For this purpose, we use MINQ (Neumaier, 1998).

The dimensionality of a TLGM for the inputs we use is too large to be solved quickly. For example, a code of  $N_d = 40$  dictionary elements that represents a 1 s spectrogram of 20 ms segments would yield a  $40 \times 50 = 2000$  dimensional problem. The Hessian matrix that is used in the quadratic programming algorithm would be a  $2000 \times 2000$  matrix which is too big for an efficient solution. We need another algorithm to find the sparse code of a TLGM. For this purpose, we use an iterative algorithm that optimizes during each iteration a subset of the code elements, which we call *Subset Selection and Quadratic Programming (SSQP)*. The pseudocode of the algorithm appears in Algorithm 1; we select a subset of  $N_{\text{set}} = 200$  elements during each iteration. Fan et al. (2005) describe a similar approach for support vector machines. Their algorithm selects a subset of two elements at a time, however, the number of elements may be more (Liao et al., 2002). Blumensath and Davies (2006) have also used a subset selection approach to find the sparse code of a high dimensional problem. An important difference between SSQP and their work is that SSQP performs several iterations of subset selection, whereas their approach selects the subset only once. We could not use such a selection criterion, as a reasonable subset would be too large to quickly calculate the Hessian matrix (size  $N_{\text{set}} \times N_{\text{set}}$ ) required by quadratic programming.

---

**Algorithm 1.** SSQP: Subset selection and quadratic programming
 

---

**Input:**  $\mathbf{a}_{\text{start}}$  and  $\Phi$   
**Output:**  $\mathbf{a}_{\text{final}}$

$q \leftarrow 0$   
 $\alpha_q \leftarrow \mathbf{a}_{\text{start}}$   
**repeat**  
 $S_{\text{active}} \leftarrow \{[i, j] | \alpha_{q[i, j]} > 0\}$   
 $S_{\text{neg}} \leftarrow \{[i, j] | \frac{\partial E}{\partial \alpha_{q[i, j]}} < 0\}$   
 $S_{\text{candidate}} \leftarrow S_{\text{active}} \cup S_{\text{neg}}$   
**sort**  $S_{\text{candidate}}$  **in descending order of**  $\left| \frac{\partial E}{\partial \alpha_{q[i, j]}} \right|$   
 $S_{\text{use}} \leftarrow \{S_{\text{candidate}}[i] | i = 1, 2, 3, \dots, N_{\text{set}}\}$   
 $\bar{\alpha}_{\text{use}} \leftarrow \{\alpha_{q[i, j]} | [i, j] \in S_{\text{use}}\}$   
 $\bar{\alpha}_{\text{new}} \leftarrow \min_{\bar{\alpha}_{\text{use}}} E(\alpha_q, \Phi)$  {using quadratic programming}  
 $\alpha_{q+1} \leftarrow \alpha_q$  **but with the**  $S_{\text{use}}$  **components replaced by**  $\bar{\alpha}_{\text{new}}$   
 $q \leftarrow q + 1$   
**until**  $E(\alpha_{q-1}, \Phi) - E(\alpha_q, \Phi) < 10^{-4}$   
 $\mathbf{a}_{\text{final}} \leftarrow \alpha_q$

---

SSQP is a gradient based algorithm which attempts to find the minimum of the quadratic error function  $E(\mathbf{a})$ .  $E$  is possibly a non-convex function. This implies that there may be several local minima. We expect that most of the code elements in the optimal solution will be zero as we are looking for a sparse code. It is therefore sensible to start the optimization with an initial guess where all the code elements are equal to zero instead of a random guess, since such a starting point is expected to be close to the optimal solution. Other large scale optimization techniques may also be used to find sparse codes (a typical problem has around 30,000 variables).

### 3.2. The dictionary

A complete dictionary has just enough elements to reconstruct any signal perfectly. There are certain benefits to using an overcomplete dictionary (a dictionary with more elements than a complete one). The code from an overcomplete dictionary compared to a complete dictionary can be less susceptible to noise, it can be more sparse and it can be more efficient in an information theoretical sense (Kreutz-Delgado et al., 2003; Lewicki and Olshausen, 1999).

How many dictionary elements constitute a complete dictionary of the TLGM? This question is equivalent to finding the minimum number of dictionary elements required to reconstruct a signal perfectly. Each dictionary element spans across all the channels, and the code can select dictionary elements to be used at any time. Therefore a signal  $\mathbf{x}$  of size  $N_c \times N_t$  can be perfectly reconstructed with a minimum of  $N_c$  dictionary elements. However, we constrain the code elements to be non-negative, so the minimum dictionary size to reconstruct any signal perfectly is  $2 \times N_c$ .

The dictionary elements of a trained dictionary could model phonemes, which are the smallest meaningful acoustic units, or it can model syllables, or even complete words. The dictionary training process is unsupervised, which means that a particular dictionary element may learn any unit of speech. In fact it would be interesting to see what type of features the trained dictionary elements model. There are 20 phonemes in the speech that make up the dataset, 11 syllables and 11 words. It is difficult to find the best dictionary size. The dictionary should at least be complete so that the code can successfully represent a signal, and the size of the dictionary may be related to the basic speech units that make up the dataset. We use a two times overcomplete dictionary, so there are 32 dictionary elements. This dictionary is large enough to capture at least all the phonemes present in the dataset.

In speech there are certain features which can be stretched over time without changing the meaning of the speech. To reflect this in the model, time scaled versions of the basic dictionary elements are constructed. We construct six scaled versions of each basic dictionary element and append them to the dictionary.

A dictionary element  $\Phi^{\{d\}}$  is scaled over time with a scaling matrix  $M_s$

$$\Phi_{\{s\}}^{\{d\}} = \Phi^{\{d\}} M_s \quad (5)$$

where  $\Phi_{\{s\}}^{\{d\}}$  is the scaled version of basic dictionary element  $d$ .  $s \in 0, 1, 2, \dots, 6$  where  $s = 0$  represents the unscaled dictionary element that spans 260 ms,  $s = 1$  corresponds to the dictionary element that is scaled to span 280 ms, etc.  $M_s$  is determined in such a way that the  $j$ th column of  $\Phi_{\{s\}}^{\{d\}}$  (which spans 20 ms) is

$$[\Phi_{\{s\}}^{\{d\}}]^{(j)} = \frac{1}{0.02} \int_{0.02 \cdot (j-1)}^{0.02 \cdot j} \Phi_{\{0\}}^{\{d\}} \left( \tau \frac{0.26}{0.26 + 0.02s} \right) d\tau \quad (6)$$

$\Phi_{\{s\}}^{\{d\}}$  is a matrix where each column is used to reconstruct a 20 ms segment of a spectrogram. We cast  $\Phi_{\{s\}}^{\{d\}}$  into the continuous time domain by letting  $\Phi_{\{s\}}^{\{d\}}(\tau) = [\Phi_{\{s\}}^{\{d\}}]^{(i)} \forall \tau \in [0.02(i-1), 0.02i]$ . Fig. 2 illustrates how a dictionary element is stretched.

The dictionary that includes the scaled elements is

$$\Phi = [\Phi_{\{0\}}, \Phi_{\{1\}}, \Phi_{\{2\}}, \dots, \Phi_{\{6\}}] \quad (7)$$

$\Phi_{\{0\}}$  is the basic dictionary which has 32 elements;  $\Phi$  therefore has  $32 \times 7 = 224$  elements.

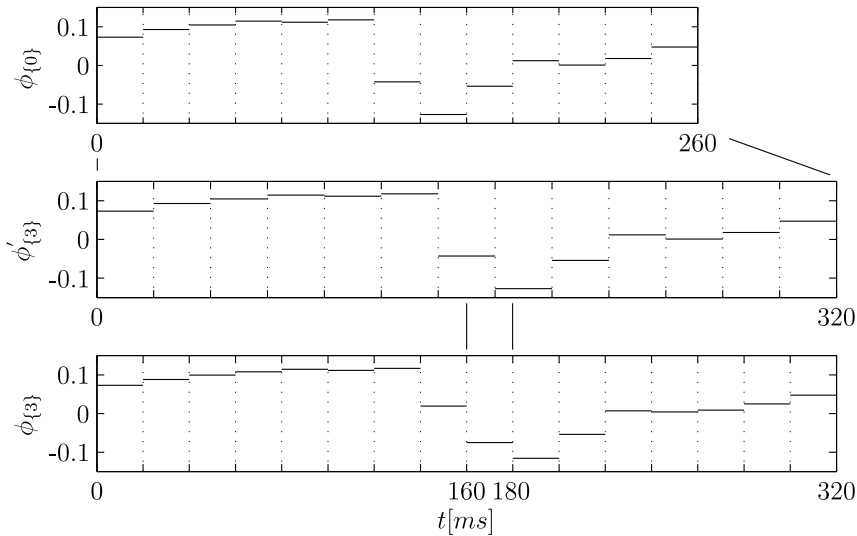


Fig. 2. An unscaled dictionary element  $\Phi_{\{0\}}^{\{d\}}$  is a  $N_c \times \Delta\Phi$  matrix. A row in the dictionary element corresponds to a channel in the spectrogram. This figure illustrates how a row of an unscaled dictionary element is scaled to 320 ms.  $\phi_{\{s\}}$  refers to a row of the  $d$ th dictionary element  $\Phi_{\{s\}}^{\{d\}}$ . The *top* plot shows a row of an unscaled dictionary element. The *middle* plot shows an intermediate step where the row is stretched to span 320 ms;  $\phi'_{\{3\}}$  still has only 13 entries. The *bottom* plot shows that  $\phi_{\{3\}}$  is created by resampling from  $\phi'_{\{3\}}$ . For example, the ninth entry which represents the interval 160–180 ms, is the time averaged integral of  $\phi'_{\{3\}}$  over that same interval. Every row of a dictionary element is stretched in the same way.

### 3.3. Training the dictionary

The code will be more efficient if the dictionary is trained to reflect the regularities of the data. We can define dictionary training as an optimization problem. The optimal dictionary will be the one that allows very sparse codes to reconstruct signals well. In other words, a sparse code  $\mathbf{a}$  will yield a small cost function  $E$  when the dictionary is optimal. The dictionary should fit the entire dataset, so we define a *total error function*  $E_T$  that is the sum of the error function of each sample  $n$  in the dataset of  $N_s$  samples

$$E_T(\mathbf{A}, \Phi) = \sum_{n=1}^{N_s} E(\mathbf{a}^n, \Phi) \quad (8)$$

with  $\mathbf{A}$  the set of sparse codes that represent the dataset.

Formally, the optimal dictionary is then

$$\Phi^* = \min_{\Phi} E_T(\mathbf{A}^*, \Phi) \quad (9)$$

with  $\mathbf{A}^*$  the set of optimal sparse codes.

The optimal dictionary is a function of the sparse codes and vice versa. The dictionary training process is therefore an iterative two-step process. First the sparse codes are determined for a given dictionary, then the dictionary is optimized for the given codes. The optimization is done by means of gradient descent with a line search. The derivative of the total error function  $E_T$  with respect to the dictionary is used as a search direction. A golden section method then finds the minimum of the error function along the search direction. The training process is exactly that of Olshausen and Field (1997), except that we use a line search, whereas they use a learning rate parameter to control the step size.

During the training process the norm of the dictionary elements tend to grow without bounds (for an explanation as to why this happens, see Olshausen and Field (1997)). We address this by forcing all dictionary elements to have a norm of one. Every dictionary element of the initial dictionary is randomized.  $\Phi_{\{s\}}^{\{d\}}$  is a random  $N_c \times \Delta\Phi$  matrix with entries sampled from a standard uniform distribution  $U(0, 1)$ .

It is computationally very expensive to train the dictionary. The reason is that it is necessary to find the sparse code of every utterance at each iteration. The SSQP algorithm can take a long time to converge, especially if the starting point is far from the final solution. For example, when we want to find the sparse code to the utterance “one one one three eight eight one”, we use a trained dictionary and set the starting point as the all-zero code  $\mathbf{a}_{\text{start}} \leftarrow \mathbf{0}$  in the SSQP algorithm. This utterance is 2.6 s long; the size of  $\mathbf{a}$  is  $224 \times 149$ ; it takes 121 iterations of SSQP to find the sparse code which has 159 non-zero elements. On an Intel Core2 1.86 GHz PC it takes 98 s for SSQP to converge.

To speed up the training process, we set the starting point of each training iteration as the solution to the previous training iteration  $\mathbf{a}_{\text{start}} \leftarrow \mathbf{a}^k$ . In this case it usually takes less than five SSQP iterations to find the sparse code. It is only for the first iteration that we set the starting point as  $\mathbf{a}_{\text{start}} \leftarrow \mathbf{0}$ .

## 4. Results related to sparse codes

To further speed up the training process we make use of two heuristics. Firstly, we start with a bigger value of  $\lambda$  than the one we would like to use; the bigger  $\lambda$ , the fewer non-zero elements are in the code and the more quickly it is computed. Secondly, we train the dictionary for the first 50 iterations only on a fifth of the samples in the dataset, since the dictionary initially does not have any structure that corresponds to the dataset – it will learn the basic structure even if the dataset is small.

The trained basic dictionary  $\Phi_{\{0\}}$  is shown in Fig. 3. The structure of the dictionary elements reveals complex features that span across frequency and time to varying degrees. It is clear that some elements are strongly localized in frequency and others more localized in time; several seem to encode specific transitions of frequency content as a function of time. The elements code sounds that span the entire width of an element (which is 260 ms).

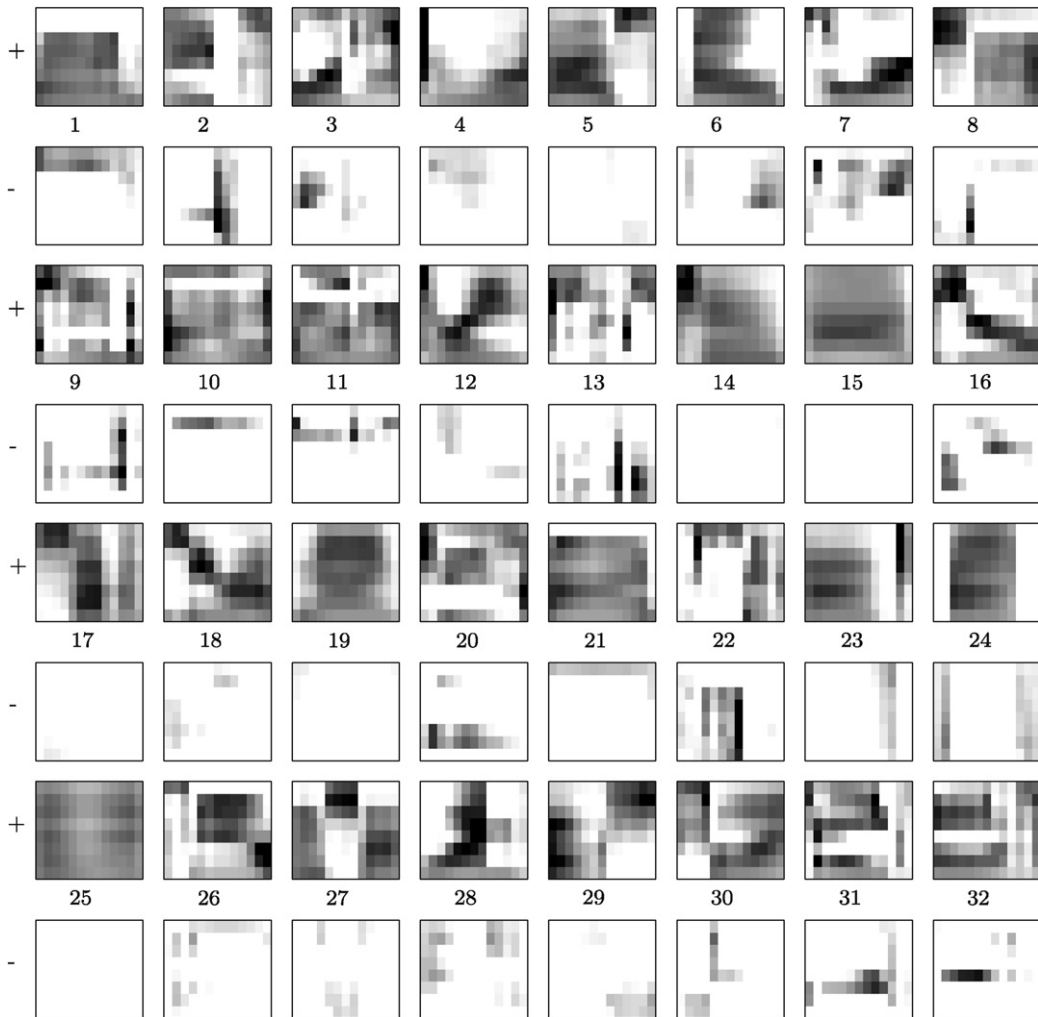


Fig. 3. This figure shows the 32 dictionary elements of the trained basic dictionary  $\Phi_{\{0\}}$ . Each element is split into positive and negative parts for better visualization. A column of a dictionary element spans 20 ms; a complete dictionary element therefore spans 260 ms. Each row in a dictionary element correlates with a channel in the spectrogram (see Fig. 1). Thus, the bottom row of a dictionary element represents signal content at 367 Hz and the top row 7321 Hz. The dictionary elements have a norm of 1. White represents values of 0 and black 0.2. Elements are numbered from left-to-right and from top-to-bottom.

We can use histograms of phoneme occurrences to form an idea of the sounds coded by the elements. A histogram is created for each phoneme  $p$  and each dictionary element  $d$ . The bins of the histogram span over time where each bin is 20 ms wide. We associate a time period  $t_b$  with each bin; it is a period that precedes a spike associated with dictionary element  $d$ . The number of entries in bin  $b$  of histogram  $(p, d)$  is the number of times phoneme  $p$  precedes a spike of dictionary element  $d$  with an offset of  $t_b$  for the entire dataset. In other words, the histogram shows how often, and at what temporal offsets, a phoneme occurs before a spike. In the TLGM a spike in the code means that a dictionary element is used to reconstruct a part of the spectrogram that *precedes* the spike. (Note that we ignore the fact that time-scaled versions of a dictionary element may in fact have been used; all the scaled versions are grouped together and are viewed as a single dictionary element.)

We use the TIMIT phoneme set (Garofolo et al., 1993) with the addition of the diphthong “ia” that occurs in “fear” and “zero”. Fig. 4 shows the histograms of phoneme occurrences for four dictionary elements. From Fig. 4a, it appears that dictionary element  $d = 2$  is used in coding the “ih-k-s” sound in “six”. It also seems that it is used to code “ey-t” which occurs in “eight”. It is not possible to conclude from this histogram alone

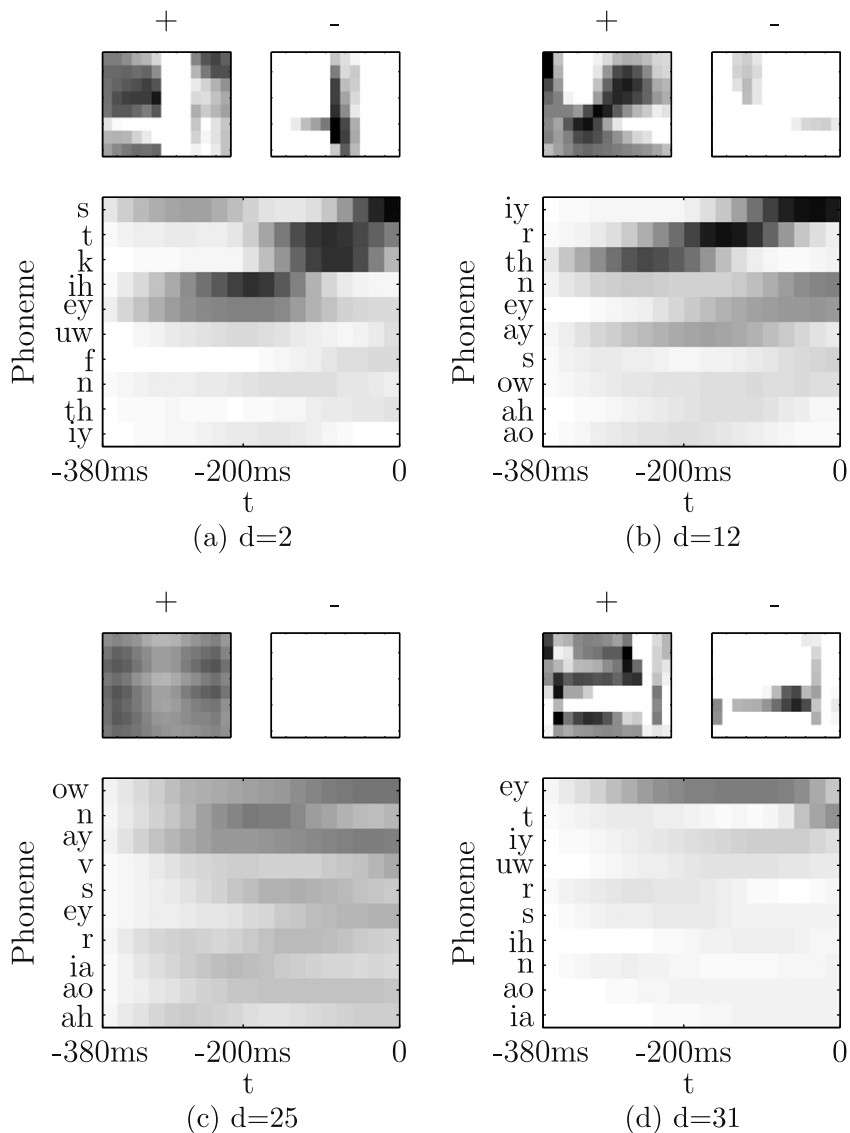


Fig. 4. The top row of each plot gives the positive and negative parts of the dictionary element. The bottom row of each plot depicts the phonemes' correlation with the given elements: it is a histogram plot that shows which phonemes preceded the spike associated with that particular model, and at what times the phoneme occurred. The histogram plot shows only the 10 phonemes that occurred most for each dictionary element. The histograms are scaled so that black represents 2000 entries in a bin.

that element  $d = 2$  actually codes the “ih” sound. The reason is that in our limited dictionary the “ih” sound always precedes the “k-s” sound. If  $d = 2$  codes only the “k-s” sound the histogram would still show that the “ih” sound often precedes a spike associated with  $d = 2$ . However, by taking a closer look at the dictionary element itself, we see that it has a period of silence between 80 and 140 ms before the spike occurs. This corresponds to the closure between the “ih” and “k” sounds, therefore it does seem that element  $d = 2$  codes the “ih” sound or at least a part of it. There are certain implications to  $d = 2$  having a large negative part. Our spectrogram transform has only positive parts, which means that  $d = 2$  cannot be used on its own. It has to be used with other dictionary elements.

Fig. 4b shows a dictionary element that codes a sound with a rising frequency. We see from the histogram that the entire word “three” often precedes a spike from  $d = 12$ . The plot of the dictionary element itself shows that it contains information to code the word “three”. The “th” fits the left part of the dictionary element; it

shows a broad distribution of energy across most of the frequencies. The right part of the dictionary element fits the rising centre-of-mass (in frequency) of the sound in “iy”.

Dictionary element  $d = 25$  (Fig. 4c) does not code a particular sound, but is rather used as a constant offset. Lastly, Fig. 4d shows that  $d = 31$  is a dictionary element with a very complex structure. The histogram shows that it is mostly used during the coding of “ey” in “eight”.

It is interesting to look at the amplitudes that spikes assume for a particular dictionary element. Fig. 5 shows histograms of the spike amplitudes for various dictionary elements. We see that the dictionary elements are not used equally often;  $d = 31$  is used less often than any of the other dictionary elements that are shown. This may be due to the complex and irregular structure of  $d = 31$ , we do not expect such a structure in speech.  $d = 25$  is used often but has a small average spike value. The spike value indicates the contribution that a dictionary element makes to reconstruct a signal. The fact that  $d = 25$  has a smaller average spike value than the other dictionary elements suggests that it too does not capture significant structure in the speech.

Note that many of the spikes have small amplitudes, regardless of the dictionary element. A spike with a small value does not contribute much to reconstructing a signal. Such a spike therefore does not signal an important event; it is rather an unfortunate feature of the way that we have formulated the sparseness function. We return to this topic in the conclusion.

Dictionary elements are not used in isolation, they are used in conjunction with other elements. It is complicated to say which acoustic unit a particular dictionary element codes as it depends on the other dictionary elements that are used in with it. We see for example from Fig. 4a that  $d = 2$  is used in coding both the words “six” and “eight” although the words do not share a phoneme.

## 5. Classification of the spike train

This section addresses the problem of decoding a spike train, i.e., how to infer or classify the spoken words encoded by the spike train.

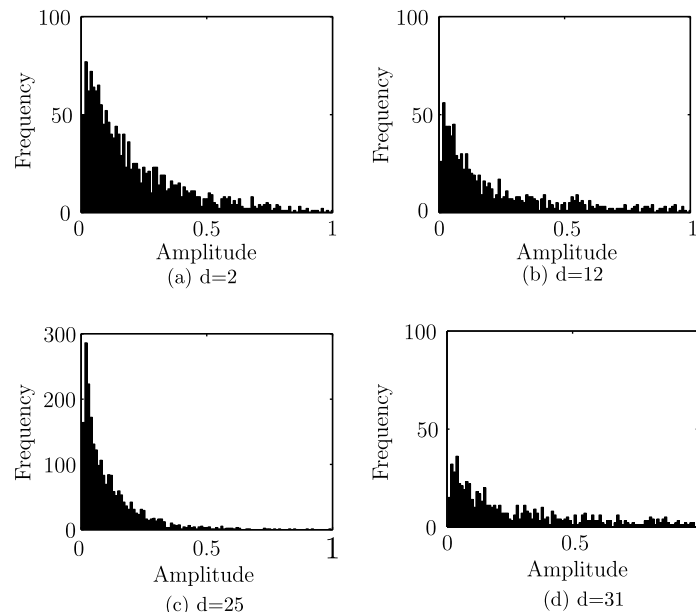


Fig. 5. The histograms of spike amplitudes for four of the dictionary elements are given. These are compiled from the sparse codes of the training set. Only nonzero spikes are included in the histograms. The histograms show that most of the spikes have very small amplitudes. We explain in the text that such spikes are not desirable. It also shows that the dictionary elements are not used equally often;  $d = 25$  is used more often than any of the other three dictionary elements.

### 5.1. Preprocessing of spike trains

As mentioned earlier, the sparse code has many elements whose amplitudes are so small that they contribute little to the reconstruction of the signal. We remove them by setting their amplitudes equal to zero. Only those spikes that have amplitudes less than a preset threshold are removed. We choose the threshold at 0.124 so that 40% of the spikes remain. We tested the classification performance against different threshold values and found that 0.124 gives the best results. The removal of small valued spikes could also be done during the process of dictionary training. We did not do that as we did not know beforehand which threshold would be best.

The spike train includes information about the temporal scaling of dictionary elements. Our motivation for using scaled dictionary elements is that some sounds may be scaled without changing the meaning of the sound. The scaling information is therefore not that important to a spike train classifier of words; what are important are the sounds that appear in the signal and the time at which they appear. We keep the spike train classifier simple by not including scaling information. It is still useful to have scaled versions of the dictionary elements, as this produces a sparse code that allows us to identify a particular sound irrespective of its duration.

The scaling information is removed by grouping spikes of the same sound together. The dictionary consists of all the scaled versions of the unscaled dictionary  $\Phi = [\Phi_{\{0\}}, \Phi_{\{1\}}, \Phi_{\{2\}}, \dots, \Phi_{\{6\}}]$ . We can split the code  $\mathbf{a}$  in the same manner;  $\mathbf{a} = [\mathbf{a}_{\{0\}}, \mathbf{a}_{\{1\}}, \mathbf{a}_{\{2\}}, \dots, \mathbf{a}_{\{6\}}]$  where  $\mathbf{a}_{\{s\}}$  refers to the sparse code associated with  $\Phi_{\{s\}}$ . The compressed code is then

$$\tilde{\mathbf{a}} = \sum_{s=0}^6 \mathbf{a}_{\{s\}} \quad (10)$$

In our case  $\tilde{\mathbf{a}}$  is a  $32 \times N_t$  matrix.

### 5.2. Classification problem

The classification problem is common in speech recognition: the most likely sequence of speech units given the features has to be determined. Once the most likely sequence is known the words in the utterance can be determined. We use the same approach as current automatic speech recognition systems to find the most likely sequence of speech units, but we have a different feature set.

This approach to the classification of a spike train requires a set of probabilistic models that fit segments of the spike train. Each model could correspond to a speech unit such as a phoneme, a triphone, a syllable, a word etc. Most of the current speech recognition systems use models that fit phonemes or triphones. The dictionary elements span 260 ms which means that a spike train model should cover speech units that are at least as long, i.e. syllables or words. We use supervised training of the spike train models and accordingly force the models to correspond to words. Supervised training is more robust than unsupervised training.

The spike train model proposed by Oram et al. (1999) is a probabilistic model of spike trains. With this model the probability  $p(\zeta_i|m)$  that segment  $i$  from the spike train  $\zeta$  fits a model  $m$  can be calculated. Accurate classification is only obtained when the models fit the data well. The last part of this section shows how the parameters of the models are adapted to fit the data.

### 5.3. The spike train model

There are a few models that can find temporal patterns in multichannel spike trains (Chi et al., 2003; Kass and Ventura, 2001; Gat and Tishby, 2001), but they do not give the probability that a spike train fits a given model.

Oram et al. (1999) proposed a spike train model that contains only the spike count (the number of spikes in a channel) and the firing rate profile (a profile of the times at which spikes are likely to occur). This model fits actual spike train data very well: once fitted to the data, it could generate spike trains that are indistinguishable from spike trains recorded in the monkey brain.

The model we use here is exactly that of Oram et al. (1999) with order statistics (Wiener and Richmond, 2003), except that we include the spike amplitude. A spike train is completely described by the times at which spikes occur  $\bar{t}$ , the amplitudes  $\bar{a}$  of those spikes and the channels  $\bar{c}$  in which they occur ( $\bar{t}, \bar{a}$  and  $\bar{c}$  are all vectors that have as many entries as the number of spikes in the spike train). The  $i$ th spike is described by its firing time  $t_i$ , amplitude  $a_i$  and channel  $c_i$ .

The probability that spike train  $\zeta$  fits model  $m$  is

$$p(\zeta|m) = p(\bar{t}, \bar{a}, \bar{c}|m) \quad (11)$$

(We force the models designated by  $m$  to correspond to words; we have three models for every word. These choices are motivated below.)

Sparseness coding assumes the activity of different code elements to be independent. Therefore,

$$p(\bar{t}, \bar{a}, \bar{c}|m) = \prod_c p(\bar{t}_c, \bar{a}_c, n_c|m, c) \quad (12)$$

with  $\bar{t}_c$  and  $\bar{a}_c$  the respective subsets of  $\bar{t}$  and  $\bar{a}$  that contain only the spikes in channel  $c$ .  $n_c$  is the spike count of channel  $c$ . There are as many channels in a spike train as there are elements in the basic dictionary; in our case the compressed code represents a spike train of 32 channels.

In order to reduce the complexity of the model, we assume that the spike times and spike amplitudes are independent of each other. Now,

$$p(\bar{t}_c, \bar{a}_c, n_c|m, c) = p_t(\bar{t}_c|m, c, n_c) p_a(\bar{a}_c|m, c) P_n(n_c|m, c) \quad (13)$$

Here  $p_t$  is the spike time probability density function. It gives the probability that the times at which spikes occur in the spike train fit the firing rate profile.  $p_a$  is the spike amplitude probability density function.  $P_n$  is the normalized spike count distribution. It gives the probability that  $n_c$  spikes will occur in a given channel.

The spike time probability density function is a function of the order statistics and the firing rate profile (Wiener and Richmond, 2003)

$$p_t(\bar{t}_c|m, c, n_c) = \prod_{h=1}^{n_c} p_t(t_{c,h}|m, c, n_c, h) \quad (14)$$

$$p_t(t_{c,h}|m, c, n_c, h) = \frac{n_c!}{(h-1)!(n_c-h)!} F_t(t_{c,h})^{h-1} f_t(t_{c,h}) [1 - F_t(t_{c,h})]^{n_c-h} \quad (15)$$

with  $f_t(t)$  the normalized spike density function and  $F_t(t)$  is its cumulative probability density.  $p_t(t_{c,h}|m, c, n_c, h)$  is the probability that for channel  $c$  of model  $m$ , the  $h$ th spike in a spike train occurs at time  $t_{c,h}$ .

The amplitudes of the spikes within a channel are also assumed to be independent so that

$$p(\zeta|m) = \prod_{c=1}^{32} \left[ P_n(n_c|m, c) \prod_{h=1}^{n_c} p_t(t_{c,h}|m, c, n_c, h) \prod_{h=1}^{n_c} p_a(a_{c,h}|m, c, h) \right] \quad (16)$$

$P_n$ ,  $p_a$  and  $f_t(t)$  are modelled with Gaussian mixture models (GMMs).  $p_a$  and  $P_n$  have each two components and  $f_t(t)$  three.

#### 5.4. Finding the most likely sequence of models

Here we show how to determine the most likely sequence of models given the spike train of an unknown utterance. From the sequence of models, we can easily determine the sequence of words as each model corresponds to a word. The spike train models can be trained in an unsupervised manner, in which case a model can represent any sound or sequence of sounds. Once the models are trained, it is necessary to determine the correspondence between sounds and models. However, supervised training is easier to conceptualize than unsupervised training. For this study, we force the models to correspond to words. Unfortunately this choice may limit the classification performance.

Part of the problem of finding the most likely sequence is segmenting the spike train. The classification problem is now to find the most likely sequence of models  $\bar{m}^*$  and segment sizes  $\bar{\Delta}^*$  given a spike train, i.e.,

$$\bar{m}^*, \bar{\Delta t}^* = \max_{\bar{m}, \bar{\Delta t}} p(\bar{m}, \bar{\Delta t} | \zeta) \quad (17)$$

We assume that each segment is independent of all others, so that using Bayes' theorem, we have

$$p(\zeta | \bar{m}, \bar{\Delta t}) = \prod_i p(\zeta_i | m_i, \Delta t_i) \quad (18)$$

where  $p(\zeta_i | m_i, \Delta t_i)$  is the probability that the  $i$ th spike segment  $\zeta_i$  is  $\Delta t_i$  long and fits model  $m_i$  (Eq. (11)). The joint probability is

$$p(\bar{m}, \bar{\Delta t}) = p_{\Delta t}(\bar{\Delta t} | \bar{m}) P(\bar{m}) \quad (19)$$

with

$$p_{\Delta t}(\bar{\Delta t} | \bar{m}) = \prod_i p_{\Delta t}(\Delta t_i | m_i) \quad (20)$$

again because we assume the segments to be independent.  $p_{\Delta t}(\Delta t_i | m_i)$  is the probability that a spike train will have a length of  $\Delta t_i$  given model  $m_i$ . It is modelled with a two-component GMM.  $P(\bar{m})$  is the language model, which for a dataset consisting of random digits and models corresponding to words is

$$P(\bar{m}) = \prod_i P(m_i) \quad (21)$$

If the models do not correspond to words, the language model would be slightly more complex. Accurate language models can be designed for a specific dataset (Rabiner and Juang, 1993). Finally, the most likely model sequence and segment sizes are

$$\bar{m}^*, \bar{\Delta t}^* = \max_{\bar{m}, \bar{\Delta t}} \prod_i p(\zeta_i | m_i, \Delta t_i) p_{\Delta t}(\Delta t_i | m_i) P(m_i) \quad (22)$$

or

$$\bar{m}^*, \bar{\Delta t}^* = \min_{\bar{m}, \bar{\Delta t}} \sum_i [-\ln p(\zeta_i | m_i, \Delta t_i) - \ln p_{\Delta t}(\Delta t_i | m_i) - \ln P(m_i)] \quad (23)$$

The above equation can be solved with dynamic programming (see for example Cormen et al., 2001), which is an efficient way to find the shortest path through a lattice.

Our set of spike train models also includes a silence model  $m_{\text{sil}}$ . This model is fixed. It has a spike count distribution

$$P_n(n_c) = \begin{cases} 1 & \text{if } n_c = 0 \\ 0 & \text{otherwise} \end{cases}$$

for every channel. The time duration of the model is set as a uniform distribution  $p(\Delta t | m = m_{\text{sil}}) = U(140 \text{ ms}, 1 \text{ s})$ . A silence period shorter than 140 ms is probably part of a word or a transition between words.

Table 1

A summary of the probability density functions (PDF) used in spike train classification

PDF	Type	Number of components
$p(\zeta   m)$	Spike train model	34 <sup>a</sup>
<i>For each channel in a spike train model, we have:</i>		
$P_n$	GMM	2
$f_i(t)$	GMM	3
$P_a$	GMM	2
<i>Associated with each spike train model there is:</i>		
$p_{\Delta t}$	GMM	2

There are three spike train models for each word and there is one silence model.

<sup>a</sup> There are three models for each of the 11 words and one silence model.

It can therefore not be labelled “silence”. We select 1 s as the upper bound as there is no silence longer than that in our dataset.  $p(\Delta t | m = m_{\text{sil}})$  could also be trained instead of being preset but it would still be necessary to limit the shortest duration that a silence can be.

A set of three models is assigned to correspond to each word in the dictionary before training starts. We use more than one model per word because the spike trains for a given word may not all be similar. It is not easy to determine the optimal number of models per word to use. Table 1 gives a summary of the probability density functions we use.

## 6. Training

In the training process the parameters of models are adapted to fit the data. The models that need training are: the spike train models with each model having three mixture models (see Table 1);  $p_{\Delta t}$  associated with each model; and also  $P(m)$ . We use expectation maximization (EM) to train the models. Firstly the expectation step finds the most likely sequence of models  $\bar{m}^*$  and segment sizes  $\bar{\Delta t}^*$  for the entire dataset, then the maximization step adapts the model parameters to increase the likelihood of  $\bar{m}^*$  and  $\bar{\Delta t}^*$ .

The expectation step uses a modified version of the standard dynamic programming algorithm used during the Viterbi search (Ostendorf et al., 1996). The modification is that our Viterbi lattice is not two dimensional; it has a third dimension, called “number of words”. A three dimensional Viterbi lattice is set up for each spike train. The three dimensions are “time”, “model number” and the “number of words” selected from start of the sequence (see Fig. 6). Paths through the lattice always point in the increasing “time” dimension. When the transition is toward a word model, it has to increase one level along the “number of words” dimension. On the other hand, if the transition is toward the silence model it does not change its “number of words” dimension.

The supervised algorithm makes some of the vertices and transitions invalid in a manner consistent with the target word sequence and the target word boundary times. Fig. 7 shows how the number of words that should be classified at specific times is bounded. From the bounds the valid vertices in the lattice are determined. At a specific time nodes along the “number of words” dimension that fall outside the bounds are invalid. The target word sequence on the other hand determines which transitions are valid: transitions from models associated with word  $w_i$  to models associated with the following word  $w_{i+1}$  are valid, as are all transition to the silence model  $m_{\text{sil}}$ .

The maximization step adapts the model parameters. As the utterances are independent of each other,  $\bar{m}_n^*$  and  $\bar{\Delta t}_n^*$  can be determined for the  $n$ th utterance independently of all the other utterances in the dataset. The maximization step maximizes the likelihood function  $\mathcal{L}(\bar{\theta})$  with

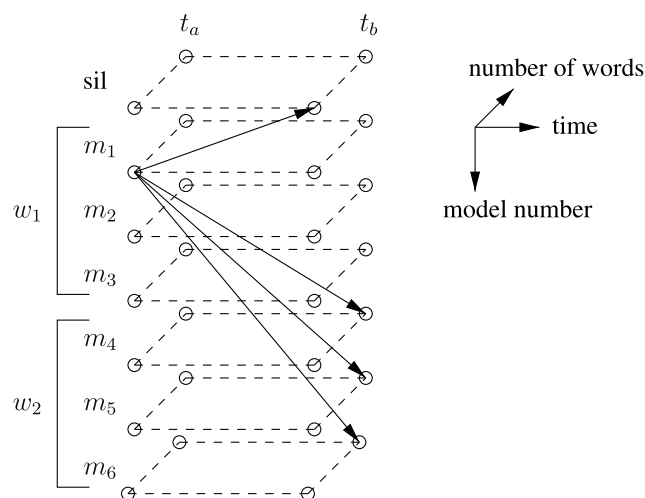


Fig. 6. From a vertex there are only two types of valid transitions to a given time instant. There is the transition to the silence model, for which the number of words since the start of the sequence do not increase; and there is the transition to all the models in the set that correspond to the next word in the sequence.

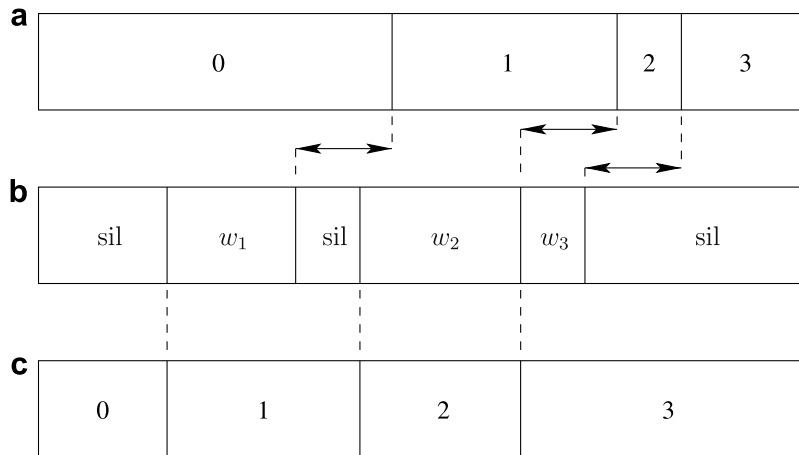


Fig. 7. This figure shows how the number of words since the start of the sequence is bounded. (b) The target sequence of words. (a) The minimum number of words that has to be in the sequence at a specific time. This is determined by the word-end boundaries. A word has to occur before the time of the last spike that can reconstruct part of that word. This time cannot be later than the word-end boundary plus the temporal length of the longest dictionary element (the elements of  $\Phi_{(6)}$  spans 380 ms). (c) The maximum number of words that can be in the sequence at a specific time is determined by the word-start boundary. A word cannot occur before its word-start boundary.

$$\mathcal{L}(\bar{\theta}) = \prod_n p(\bar{m}_n^*, \bar{\Delta t}_n^* | \zeta_n, \bar{\theta}) \tag{24}$$

where the subscript  $n$  refers to the  $n$ th utterance in the data set and  $\bar{\theta}$  refers to the model parameters.

The problem is more readily solved by minimizing the negative log-likelihood function

$$\bar{\theta}^* = \min_{\bar{\theta}} - \sum_n \ln p(\bar{m}_n^*, \bar{\Delta t}_n^* | \zeta_n, \bar{\theta}) \tag{25}$$

This can efficiently be done by taking the derivative of the negative log-likelihood function and finding the parameters for which the derivative is equal to zero. These derivatives are available in most text books that discuss GMMs and also in Juang et al. (1986).

The expectation step and the maximization step are repeated one after the other until the model parameters have converged.

Table 2  
The confusion matrix of classification of the test set

Recognized class	True class										
	One	Two	Three	Four	Five	Six	Seven	Eight	Nine	Zero	Oh
One	240	2	2	14	4	0	0	9	12	1	8
Two	0	266	5	2	0	1	5	7	3	3	1
Three	2	9	283	1	0	0	0	13	0	1	0
Four	7	4	1	260	5	1	0	0	0	0	3
Five	2	3	2	11	230	1	3	1	8	2	19
Six	1	3	1	0	1	288	2	0	0	0	0
Seven	0	10	2	2	1	8	259	1	0	12	1
Eight	4	5	9	0	3	3	0	251	5	1	2
Nine	19	2	5	2	4	0	0	7	211	4	14
Zero	0	5	2	1	1	0	1	0	1	280	1
Oh	4	15	1	7	8	0	0	2	6	1	221

The diagonal entries are words that are correctly classified; the off-diagonal entries are words that have been replaced by incorrect words during classification.

### 6.1. Initialization of the spike train models

The likelihood function has many local minima and EM finds one of them. The starting point for EM has a great influence on the quality of the solution. The solution that EM finds is much better when the starting point is in a region of a good solution, than when a random starting point is used.

The very first iteration of EM uses the starting point  $\hat{\theta}_{\text{start}}$  to determine the most likely model sequence  $\bar{m}^*$  and segment sizes  $\bar{\Delta t}^*$ . It then adapts the model parameters to better fit  $\bar{m}^*$  and  $\bar{\Delta t}^*$ . We do not know what a good starting point should be, so we skip the first step of the first iteration of EM and manually set  $\bar{m}^*$  and  $\bar{\Delta t}^*$ .  $\bar{\Delta t}^*$  is set according to the dataset labels.  $\bar{m}^*$  is set by dividing all the samples of a word evenly between the models of that word. For example, the first occurrence of “three” in the dataset is labelled as  $m = 7$ , the second

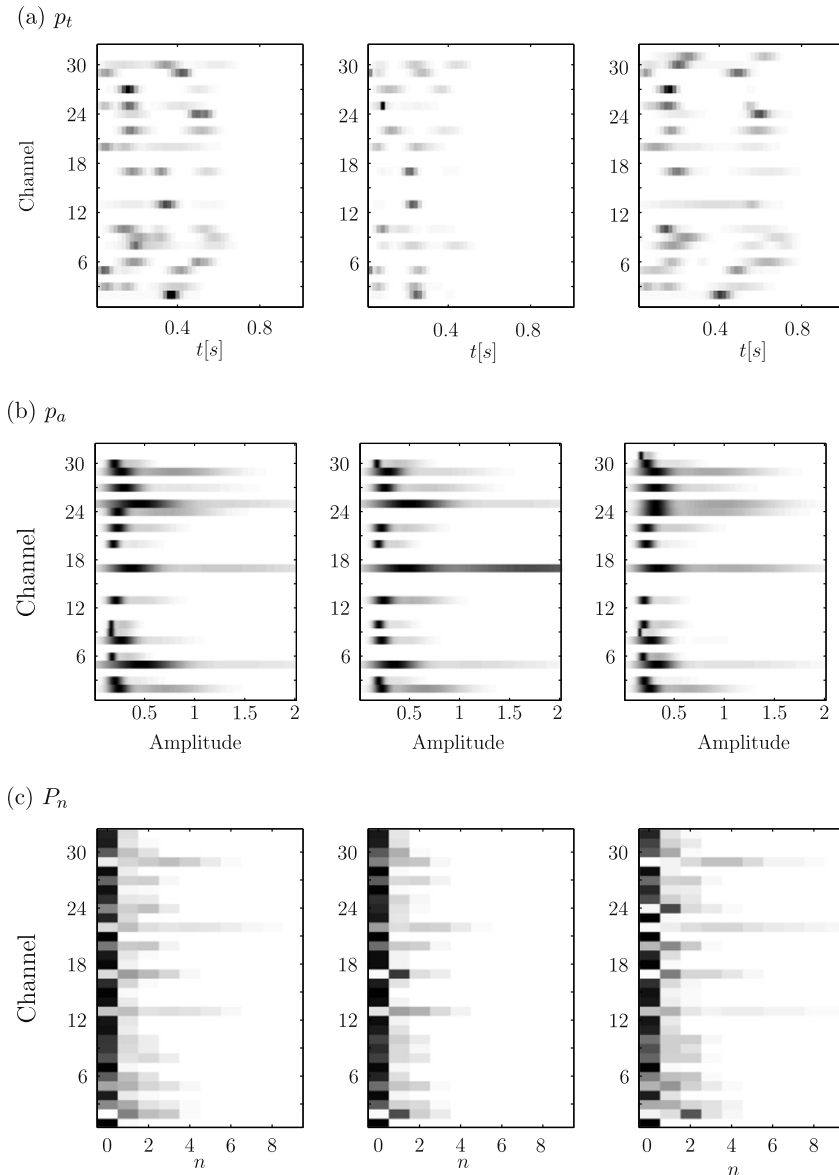


Fig. 8. The plots shows the components of the three models that are associated with the word ‘six’. From the left column to the right the models are  $m = 16$ ,  $m = 17$  and  $m = 18$ , respectively. Plots (a), (b) and (c) show  $p_t$ ,  $p_a$  and  $P_n$  for each respective model. For (a) and (b), we show only those channels that are highly active, i.e. only the channels for which  $P_n(0) \leq 0.2$ .

occurrence as  $m = 8$ , the third as  $m = 9$ , the fourth again as  $m = 7$ , etc. This initialization works well, as all the models are forced to be in a region of good solutions.

## 7. Results related to spike train classification

In the EM training process the spike train models converged to a stable solution in fewer than ten iterations. The performance on the test set (1000 utterances) is a word error rate (WER) of 19%. Out of the 3222 words in the test set, there are 65 deletions, 178 insertions and 368 replacements. Table 2 shows the confusion matrix of the replacements. The majority of confusions are predictable from phonetic similarities – for example, “one” and “nine” are often confused because of their common terminal nasals and confusable initial phonemes.

Fig. 8 shows the components  $p_t$ ,  $p_a$  and  $P_n$  for the three models that code the word “six” (they are  $m = 16$ ,  $m = 17$  and  $m = 18$ ). From the plot, we see that  $P_n(0)$  for channel 2 of every model is very small. This shows that dictionary element  $d = 2$  is used often in the reconstruction of “six”.  $m = 16$  has a very similar  $p_t$  to  $m = 18$ . There is an observable difference between the  $P_n$  of these two models, but the difference is not significant. It appears that the EM algorithm was trapped in a local minimum where these two models are close together.

Fig. 9 shows how often each model is used ( $P(m)$ ) as well as the lengths of spike trains that are coded by each model ( $p_{\Delta t}(\Delta t|m)$ ). We see from  $p_{\Delta t}(\Delta t|16)$  and  $p_{\Delta t}(\Delta t|18)$  that the models will mostly be selected when the time span  $\Delta t$  is rather long. The time span is in fact much longer than it takes to say the word “six”. Inspection of the results reveal that this is because these models are mostly selected when “six” is the last word in an utterance, or when “six” is followed by a period of silence. If the models had been better trained,  $p_{\Delta t}(\Delta t|m)$  would have had a higher probability that the lengths correspond to the time it actually takes to say “six”. The expectation step in the EM algorithm would then use the silence model  $m_{\text{sil}}$  to take up the periods of silence following “six”. This suggests that the EM algorithm became stuck in a non-optimal local extremum. We see from  $P(16)$  and  $P(18)$  in Fig. 9 that these models are not used often, which agrees with the fact that they are mostly used at the end of utterances or when long periods of silence follows “six”.

A comparison between the performance of this sparse coding system and other systems for continuous speech recognition gives an indication of the effectiveness of the sparse coding approach. We compared the performance of the current system to that of Hidden Markov Model (HMM)-based speech recognition on the same spectrogram representations we use. HMMs are typically used with features such as Mel-frequency cepstral coefficients, which are approximately independent. Employing HMMs with features directly extracted from spectrograms requires some care, since the features will tend to be correlated to a significant degree. We were able to deal with this issue by using components with full covariance matrices. Recognition accuracy was

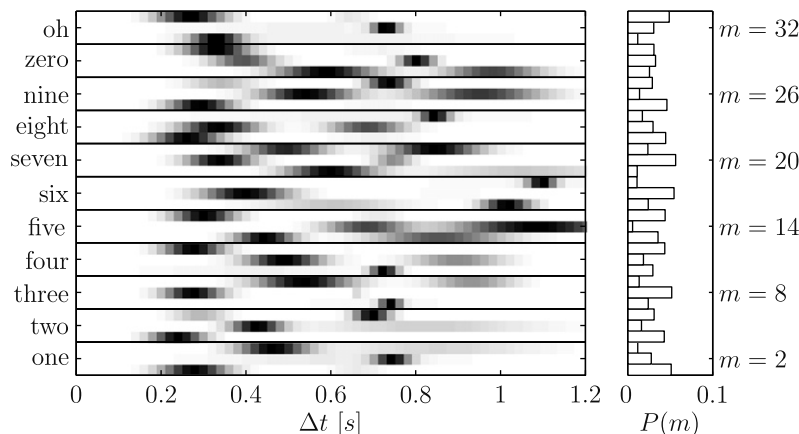


Fig. 9. The left figure gives  $p_{\Delta t}(\Delta t|m)$ ; the right figure gives  $P(m)$ . There are 33 rows in  $p_{\Delta t}(\Delta t|m)$  and in  $P(m)$ , each row corresponds to a model. For example  $p_{\Delta t}(\Delta t|1)$  and  $P(1)$  appear in the bottom row of each plot while models  $m = 7-9$  are associated with the word “three”.

not affected to a significant degree when we used more than one mixture component. The comparison could also be made with methods such as Dynamic Time Warping with full covariance matrices, but the complexities of developing such a system for speaker-independent recognition were not considered justified for our rather preliminary comparison.

The HMM system achieved a WER of 15% for models based on monophones. The models were single mixture components with full covariance matrices. Straightforward HMM-based speech recognition with triphones achieves a 3% WER on the TIDIGITS dataset when Mel-frequency cepstral coefficients (MFCCs) are used instead of the coarse spectrograms we use. The WER of 15% for the HMM system is slightly better than the 19% WER that sparse coding achieves. We return to these performance differences below.

## 8. Conclusion

We have shown how continuous speech recognition of a small dataset can be done with sparse coding and spike train classification. The results are promising: for the input features used here, the performance of speech recognition based on sparse coding is comparable to HMM-based speech recognition. These results are, however, not competitive with those achievable using state-of-the-art features; incorporating such features into our system would not have been feasible for computational reasons.

There are many areas of the sparse coding approach that need to be investigated in order to determine whether these methods can be competitive with current state-of-the-art systems. Most obviously, other feature sets such as spectrograms with a greater number of channels, or even MFCCs should be investigated. It should be noted that we do not require the MFCC coefficients to be sparse – as long as they are relatively stable in a particular acoustic context, our coding algorithm will extract a sparse set of events that describe the non-sparse MFCC coefficients. The training process however requires significantly more efficient training algorithms when the size of the input representation increases.

The size of the dictionary plays an important role in the properties of sparse codes. If the dictionary size is increased while  $\lambda$  remains constant, the dictionary elements will code more complex features. For a dictionary that has too many elements, the features will not be general enough to capture the underlying structure in the signal. This will cause the classification performance to suffer. We did not study the effect the size of the dictionary has on the classification performance or on the type of features that the dictionary elements code. Here too more work needs to be done.

The effect of  $\lambda$  on the performance needs to be investigated. We found that for a 32 element dictionary, the performance decreases for  $\lambda > 0.3$ , but we did not check the performance for  $\lambda < 0.3$ .

The sparseness function we use causes the sparse code to have many components with small amplitudes. These components are not desirable as they do not convey important information about the signal. We found that by removing 40% of the spikes the performance improves significantly. Along this line there are two alternatives to the approach we followed here. The spikes could be removed during the dictionary training process, i.e. the dictionary is trained on a code whose small code elements are removed. This will change the dictionary elements of the trained dictionary, which in turn may improve the classification performance of the system. The second alternative is to use a sparseness function that is not only a function of the code element amplitudes but also the activity of code elements. For example,

$$S(a) = \begin{cases} 0 & \text{if } a = 0 \\ 1 + a - \frac{\beta}{2} a^2 & \text{if } a > 0 \end{cases}$$

will produce a code with fewer elements close to zero. Such sparseness functions cause the optimization problem to become non-quadratic and computationally much more expensive. They also make the derivative of the error function undefined at  $a = 0$ . The two alternatives mentioned are unfortunately mutually exclusive; the first is easier to implement in the current framework, whereas the second would require different algorithms to find sparse codes as the derivative cannot be used at  $a = 0$ .

The problem of finding a sparse code is not yet satisfactorily solved. We use an iterative optimization approach that is computationally expensive. Basis selection methods may be more efficient. However, we

found that *matching pursuit* (Mallet and Zhang, 1993) yields sparse codes that are not robust; two utterances that sound very similar produce sparse codes that are not similar at all (results not shown).

The classification of spike trains is done on trains that do not include the scaling of dictionary elements in any way. The performance could improve when scaling information is used. This can be done by simply adding another dimension to a spike train, so that a given spike is completely described by its scale  $s$ , firing time  $t$ , amplitude  $a$  and channel  $c$ .

We have forced the spike train models to correspond to words, while most modern HMM-based speech recognition systems model monophones or triphones. When it comes to discovering the basic units of speech, unsupervised training of the spike train models can be very powerful. With unsupervised training it is possible to discover those units of speech that are statistically significant. Hopefully, they will correlate with the basic units that the brain uses. Another advantage of unsupervised training is that the number of models per class does not have to be specified, only the number of spike train models. In practical applications, it will always be necessary to use supervised training on some part of a classification system; if it is not done on the spike train models it should be done on a next level of processing.

The spike train model of Oram et al. (1999) with order statistics (Wiener and Richmond, 2003) appears to be a useful model to use for spike train classification. The system's performance is probably capped by the sparse coding process and not by the spike train classifier.

The results obtained in our investigation suggest that the further study of the mentioned issues will be a worthwhile endeavor.

## Acknowledgements

It is a pleasure to thank two anonymous reviewers for several useful insights on presentation, related work and conceptual matters.

## References

- Allen, J.B., 1994. How do humans process and recognize speech?. *IEEE Transactions on Speech and Audio Processing* 2 (4) 567–577.
- Blumensath, T., Davies, M., 2006. Sparse and shift-invariant representation of music. *IEEE Transactions on Audio, Speech, and Language Processing* 14 (1), 50–57.
- Cambridge University, Engineering Department, 2006. Hidden Markov Model Toolkit version 3.4. <<http://htk.eng.cam.ac.uk>>.
- Chi, Z., Rauske, P.L., Margoliash, D., 2003. Detection of spike patterns using pattern filtering, with applications to sleep replay. *Neurocomputing* 52–54, 19–24.
- Cho, Y.C., Choi, S., 2005. Nonnegative features of spectro-temporal sounds for classification. *Pattern Recognition Letters* 26, 1327–1336.
- Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C., 2001. *Introduction to Algorithms*, second ed. MIT Press and McGraw-Hill.
- Fan, R.E., Chen, P.H., Lin, C.J., 2005. Working set selection using second order information for training support vector machines. *Journal of Machine Learning Research* 6, 1889–1918.
- Földiák, P., Young, M., 1995. Sparse coding in the primate cortex. In: Arbib, M. (Ed.), *The Handbook of Brain Theory and Neural Networks*. MIT Press, pp. 895–898.
- Garofolo, J.S., Lamel, L.F., Fisher, W.M., Fiscus, J.G., Pallett, D.S., Dahlgren, N.L., Zue, V., 1993. *TIMIT Acoustic-Phonetic Continuous Speech Corpus*. Linguistic Data Consortium, Philadelphia.
- Gat, I., Tishby, N., 2001. Spotting neural spike patterns using an adversary background model. *Neural Computation* 13, 2681–2708.
- Hermansky, H., 1990. Perceptual linear predictive (PLP) analysis of speech. *Journal of the Acoustical Society of America* 87, 1738–1752.
- Hermansky, H., 1998. Should recognizers have ears? *Speech Communication* 25, 3–27.
- Hermansky, H., Morgan, N., 1994. Rasta processing of speech. *IEEE Transactions on Speech and Audio Processing* 2 (4), 578–589.
- Holmberg, M., Gelbart, D., Ramacher, U., Hemmert, W., 2005. Automatic speech recognition with neural spike trains. In: *Interspeech 2005 – Eurospeech, Proceedings of the Ninth European Conference on Speech Communication and Technology*.
- Huggins, A.W.F., 1975. Temporally segmented speech. *Perception and Psychophysics* 18 (2), 149–157.
- Juang, B.H., Levinson, S.E., Sondhi, M.M., 1986. Maximum likelihood estimation for multivariate mixture observations of Markov chains. *IEEE Transactions on Information Theory* 32 (2), 307–309.
- Kass, R.E., Ventura, V., 2001. A spike-train probability model. *Neural Computation* 13, 1713–1720.
- Kral, A., 2000. Temporal code and speech. *Acta Otolaryngologica* 120, 529–530.
- Kreutz-Delgado, K., Murray, J.F., Rao, B.D., Engan, K., Lee, T.W., Sejnowski, T.J., 2003. Dictionary learning algorithms for sparse representation. *Neural Computation* 15, 349–396.
- Kwon, O.W., Lee, T.W., 2004. Phoneme recognition using ICA-based feature extraction and transformation. *Signal Processing* 84 (6), 1005–1019.
- Leonard, R.G., Doddington, G., 1993. *TIDIGITS*. Linguistic Data Consortium, Philadelphia.

- Lewicki, M.S., 2002. Efficient coding of natural sounds. *Nature Neuroscience* 5 (4), 356–363.
- Lewicki, M.S., Olshausen, B.A., 1999. Probabilistic framework for the adaptation and comparison of image codes. *Journal of the Optical Society of America* 16 (7), 1587–1601.
- Liao, S.P., Lin, H.T., Lin, C.J., 2002. A note on the decomposition methods for support vector regression. *Neural Computation* 14, 1267–1281.
- Loiselle, S., Rouat, J., Pressnitzer, D., Thorpe, S., 2005. Exploration of rank order coding with spiking neural networks for speech recognition. In: *International Joint Conference on Neural Networks*, Montreal, Canada.
- Mallet, S., Zhang, Z., 1993. Matching pursuits with time–frequency dictionaries. *IEEE Transactions on Signal Processing* 41 (12), 3397–3415.
- Massaro, D.W., 1972. Preperceptual images, processing time and perceptual units in auditory perception. *Psychological Review* 79 (2), 124–145.
- Mercier, D., Séguier, R., 2002. Spiking neurons (stanns) in speech recognition. In: *Proceedings of the Third WSES International Conference on Neural Networks and Applications*, Interlaken.
- Moller, A., 1999. Review of the roles of temporal and place coding of frequency in speech discrimination. *Acta Otolaryngology* 119, 424–430.
- Näger, C., Storck, J., Deco, G., 2002. Speech recognition with spiking neurons and dynamic synapses: a model motivated by the human auditory pathway. *Neurocomputing* 44–46, 937–942.
- Neumaier, A., 1998. MINQ – general definite and bound constrained indefinite quadratic programming. <<http://www.mat.univie.ac.at/neum/software/minq/>>.
- Olshausen, B.A., 2002. Sparse codes and spikes. In: Rao, R.P.N., Olshausen, B.A., Lewicki, M.S. (Eds.), *Probabilistic Models of the Brain: Perception and Neural Function*. MIT Press, pp. 257–272.
- Olshausen, B.A., Field, D.J., 1997. Sparse coding with an overcomplete basis set: a strategy employed by V1? *Vision Research* 37, 3311–3325.
- Oram, M.W., Wiener, M.C., Lestienne, R., Richmond, B.J., 1999. Stochastic nature of precisely timed spike patterns in the visual system neuronal responses. *Journal of Neurophysiology* 81, 3021–3033.
- Ostendorf, M., Digilakis, V.V., Kimball, O.A., 1996. From HMMs to segment models: a unified view of stochastic modeling for speech recognition. *IEEE Transactions on Speech and Audio Processing* 4, 360–378.
- Rabiner, L., Juang, B.H., 1993. *Fundamentals of Speech Recognition*. Prentice Hall.
- Shannon, R.V., Zeng, F.G., Kamath, V., Wygonski, J., Ekelid, M., 1995. Speech recognition with primarily temporal cues. *Science* 270 (5234), 303–304.
- Smith, E., Lewicki, M.S., 2005. Efficient coding of time-relative structure using spikes. *Neural Computation* 17, 19–45.
- Verstraeten, D., Schrauwen, B., Stroobandt, D., Campenhout, J.V., 2005. Isolated word recognition with the liquid state machine: a case study. *Information Processing Letters* 95, 521–528.
- Vinje, W.E., Gallant, J.L., 2000. Sparse coding and decorrelation in primary visual cortex during natural vision. *Science* 287.
- Wang, K., Shamma, S.A., 1995. Spectral shape analysis in the central auditory system. *IEEE Transactions on Speech and Audio Processing* 3 (5), 382–395.
- Wiener, M.C., Richmond, B.J., 2003. Decoding spike trains instant by instant using order statistics and the mixture-of-Poissons model. *Journal of Neuroscience* 23 (6), 2394–2406.