CHAPTER THREE

SPIKE TRAIN CLASSIFICATION

The previous chapter shows how a speech signal can be transformed into a multichannel or multineural spike train. In this chapter we will address the problem of decoding the spike train, i.e. how to infer or classify the spoken words encoded by the spike train.

A spike train can be decoded when the way in which spikes carry information is fully understood. For example, does the specific time at which a spike occur carry any meaning, or is the information contained in the average firing rate of a neuron? We first look at a few of the coding schemes proposed for spike trains. After that we present a probabilistic model for spike trains and we show how to use this model to infer the words that are encoded by a spike train.

3.1 CODING SCHEMES

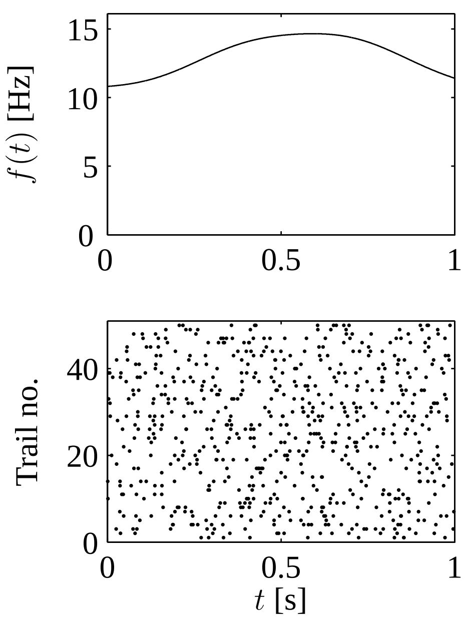
In order to decode a spike train, it is necessary to understand how information is encoded by the spike train. We do not understand the neural code completely. However recent advances in the technology of neuroscience have made it possible to form a better picture of the neural code.

The neural code represents information about the external world. Neurons very close to the senses carry the most information about stimuli. Subsequent stages in the neural system transform the information and discard irrelevant information. The purpose of processing sensory information is to extract features in the stimulus relevant to behaviour.

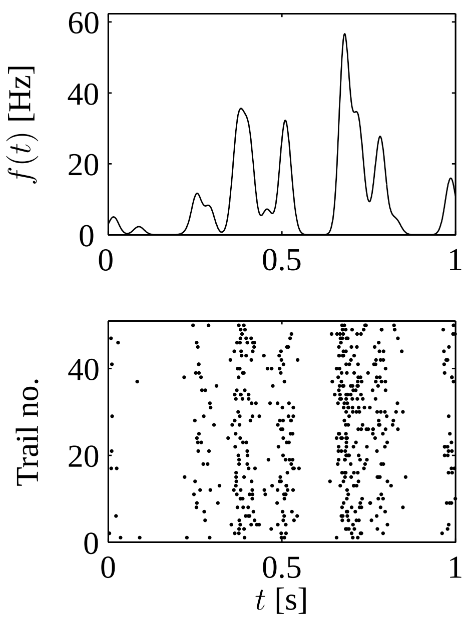
Originally it was thought that information is encoded only in the *firing rate* of neurons. This led to the development of the very successful multi-layer perceptron. The numerical value of a perceptron represents the firing rate of a neuron; the sigmoid activation function corresponds to the limited range of firing rates that a real neuron can achieve. Spike train models that are based on the firing rate view often make use of a Poisson model for which the rate parameter is stimulus dependent.

Another view of the neural code, the *temporal coding* view, is that the precise timing of a spike also carries information in addition to the firing rate (Dayan and Abbott, 2001). It is difficult to verify experimentally which coding scheme fits the neural code best. Figure 3.1 shows two time-varying stimuli and the response of a neuron to those stimuli. The neuron is a stochastic model with the probability that a spike occur proportional to the firing rate. In figure 3.1(a) there is no temporal pattern in the spikes that code the slowly varying time stimulus. The information is coded in the firing rate of a neuron measured over a rather long time frame. In figure 3.1(b) it appears as if there is a pattern in the spike train that may contain additional information. However this is not the case, the patterns are the result of a quickly varying stimulus. The firing rate of a quickly varying stimulus should be decoded from the spike train by taking the average number of spikes in a short time frame. We see that the difference between rate coding and temporal coding is more than a matter of time scales. A code cannot be called a temporal code only because there are patterns in the spike train that occur often.

Oram, Wiener, Lestienne and Richmond (1999) has proposed a *spike count-matched model*. The model is based on the *firing rate* profile (firing rate as a function of time) and the *spike count* (number of spikes in response to a stimulus). This model fits actual spike train



(a)



(b)

Figure 3.1: The response of a stochastic neuron to a time-varying stimulus is shown in each figure. The experiment is repeated for 50 trials. (a) There is no pattern in the spike train of the neuron in response to a slowly varying stimulus. (b) There is an obvious pattern in the spike train of a neuron in response to the quickly varying stimulus, but this is not enough reason to assume that it is a temporal code.

data from the lateral geniculate nucleus, the primary visual cortex (V1), and primary motor cortex of a monkey. They show that the precise patterns in spike trains are directly related to the firing rate modulation and the spike count distribution. It is therefore not a temporal coding model. The precise timing of spikes does not carry information beyond the firing rate in the three mentioned areas of the monkey brain.

There are also other coding schemes that fit neural data. The *time-to-first-spike* scheme considers only the time from stimulus onset to the first spike. This scheme fits real data from certain tasks very well; tasks that require a quick response allow only time for neurons in the different processing stages to fire a single spike. For example, faces are recognized by humans within 150ms (Thorpe, Fize and Marlot, 1996), just enough time for each stage to fire a single spike. This scheme is just a special case of the spike count-matched model. Another coding scheme is *synchrony*. The fact that certain neurons spike at almost the same time carries information. Again this is a special case of the spike count-matched model that is extended to include multiple neurons.

The spike count-matched model fits *single neuron* data from three areas of the monkey brain. The model should be expanded and be tested on multineural spike trains. The discussion between neuroscientists on the actual neural coding scheme is not yet over. More data needs to be analyzed to find the true coding scheme. It seems as if different areas of the brain may use different coding schemes.

3.2 CLASSIFICATION OF THE SPIKE TRAIN

Accurate classification of spike trains requires that patterns in the spike trains correlate well with the words in the utterance. For example, if two different people say the same words, we would ideally like the spike trains that code those utterances to be identical. This is seldom possible because people speak at different rates, at a different pitch etc. Even if spike the trains that correspond to the same word are not identical, it is still possible to classify spike trains accurately. This is possible when the spike trains of the same word but different utterances are similar enough, and when the spike trains of different words are dissimilar enough.

A spike train can be classified by matching patterns in a spike train with templates. The templates could correspond to a specific part of a word, for example a phoneme, a triphone, a syllable or even a complete word. There are a few models that can find temporal patterns in multichannel spike trains (Chi, Rauske and Margoliash, 2003; Kass and Ventura, 2001; Gat and Tishby, 2001); they do not give the probability that a spike train fits a model.

Current speech recognition systems use a powerful probabilistic approach to classification: the most likely sequence of words given the features is determined. There are well established methods for finding the model parameters that will lead to good classification for the probabilistic approach. We use the same classification approach as current automatic speech recognition systems, but we have a different feature set.

The spike count-matched model with order statistics (Wiener and Richmond, 2003) is well suited to this problem. With this model the probability $p(\zeta_i \mid m)$ that segment i from the spike train $\boldsymbol{\zeta}$ fits a model m can be calculated and is used to find the most likely sequence of models. The model can easily be extended to include multichannel spike trains; this extended model is the spike train model we use. We use supervised training of the models and accordingly force them to correspond to words. This choice is discussed below in more detail.

3.2.1 THE SPIKE TRAIN MODEL

A spike train is completely described by the times $\bar{t}$ at which spikes occur, the amplitudes $\bar{a}$ of those spikes and the channels $\bar{c}$ in which they occur. The ith spike is described by its firing time t_i, amplitude a_i and channel c_i.

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

$$p(\boldsymbol{\zeta} \mid m) = p(\bar{t}, \bar{a}, \bar{c} \mid m). \tag{3.1}$$

In section 2.6.2 we showed that the sparseness function assumes the activity of different code elements to be independent. Therefor

$$p(\bar{t}, \bar{a}, \bar{c} \mid m) = \prod_{c} p(\bar{t}_c, \bar{a}_c, n_c \mid m, c) \tag{3.2}$$

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.

In order to reduce the complexity of the model, we assume that the spike times and spike amplitudes are independent of each other. It is reasonable to assume that the spike times are independent. Figure 2.17 shows that there is a weak correlation between spike times, but the correlation exists because it codes the same word. So

$$p(\bar{t}_c, \bar{a}_c, n_c \mid m, c) = p_t(\bar{t}_c \mid m, c, n_c) p_a(\bar{a}_c \mid m, c) P_n(n_c \mid m, c) \tag{3.3}$$

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 than 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)

$$p_t(\bar{t}_c \mid m, c, n_c) = \prod_h p_t(t_{c,h} \mid m, c, n_c, h) \tag{3.4}$$

$$p_t(t_{c,h} \mid m, c, n_c, h) = \frac{n_c!}{(h-1)!(n_c-h)!} F_t(t_{c,h})^{h-1} f_t(t_{c,h}) [1 - F_t(t_{c,h})]^{n_c - h} \tag{3.5}$$

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

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

$$p(\boldsymbol{\zeta} \mid m) = \prod_c \left[P_n(n_c \mid m, c) \prod_h p_t(t_{c,h} \mid m, c, n_c, h) \prod_h p_a(a_{c,h} \mid m, c, h) \right] \tag{3.6}$$

p_a and $f_t(t)$ are modelled with Gaussian mixture models (GMMs). p_a has two components and $f_t(t)$ three. P_n is modelled with a discrete GMM having two components.

3.2.2 AN EXAMPLE OF SPIKE TRAIN MODELS

Figure 3.2(a)-(c) shows three simplified spike train models. The models do not use spike amplitude information. One gets a good idea of how spike train models are able to classify a spike train by comparing a few models. Consider for example models $m = 1$ and $m = 3$. Their spike count distributions are similar except for channel 4. The probability that model $m = 3$ does not contain a spike in channel 4 is much lower than the probability that model $m = 1$ does not contain a spike in that channel. By just counting the number of spikes in channel 4 of an unclassified spike train, we have a good indication whether that spike train was generated by model $m = 1$ or model $m = 3$.

The number of spikes in channel 4 would not be a good way to classify spike trains generated by models $m = 2$ and $m = 3$. The spike count distributions of these two models are very similar for channel 4. In this case it will be better to use the spike times of spikes in channel 3. By inspecting all three models we see that if a random spike train is generated with equal probability $P(m) = 1/3$ by any of the models, then we will be able to predict very accurately which model generated that spike train.

Figure 3.2(d) shows a spike train that is generated by model $m = 1$. The spike train will be correctly classified as it fits model $m = 1$ the best.

3.2.3 FINDING THE MOST LIKELY SEQUENCE OF MODELS

Here we show how to determine the most likely sequence of models given an unknown spike train. 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 $\overline{m}^*$ and segment sizes $\overline{\Delta t}^*$ given a spike train, i.e.

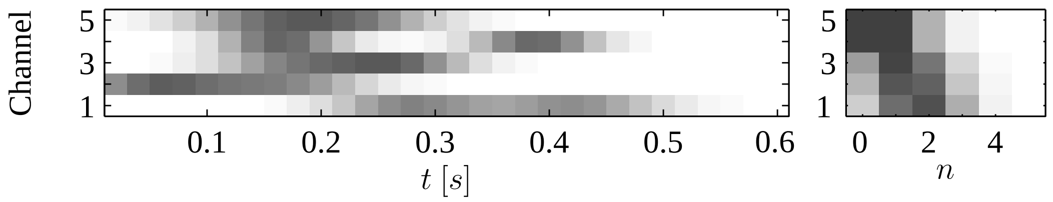
$$\overline{m}^*, \overline{\Delta t}^* = \max_{\overline{m},\overline{\Delta t}} p(\overline{m}, \overline{\Delta t} \mid \boldsymbol{\zeta}) \tag{3.7}$$

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

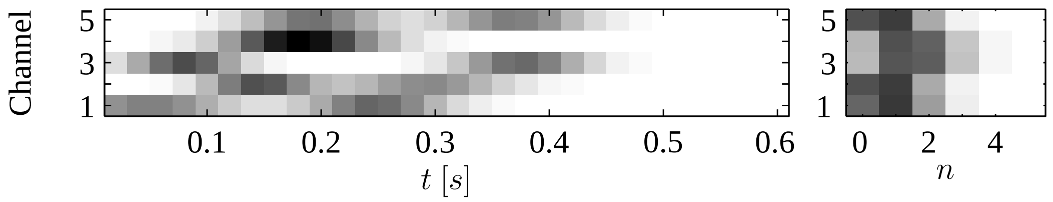
$$p(\boldsymbol{\zeta} \mid \overline{m}, \overline{\Delta t}) = \prod_i p(\zeta_i \mid m_i, \Delta t_i) \tag{3.8}$$

where $p(\zeta_i \mid m_i, \Delta t_i)$ is the probability that the ith spike segment ζ_i is Δt_i long and fits model m_i (equation 3.1). The joint probability is

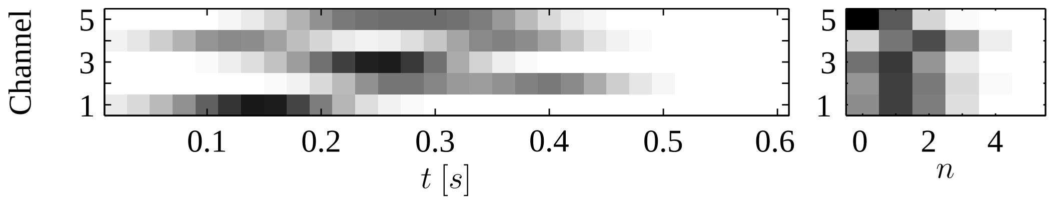
$$p(\overline{m}, \overline{\Delta t}) = p_{\Delta t}(\overline{\Delta t} \mid \overline{m}) P(\overline{m}) \tag{3.9}$$



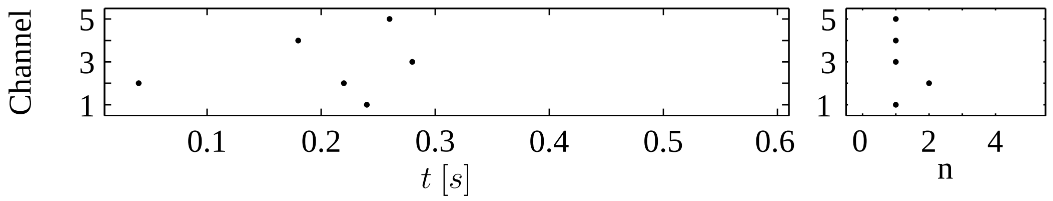
(a) m=1



(b) m=2



(c) m=3



(d)

Figure 3.2: The left column of (a) to (c) gives the normalized spike density function f_t, while the right column gives the spike count distribution P_n. (d) shows a spike train that best fits model $m = 1$. In fact $\ln(p(\boldsymbol{\zeta} \mid m))$ equals for models (a), (b) and (c) respectively -22.3, -29.6 and -36.9.

with

$$p_{\Delta t}(\overline{\Delta t} \mid \overline{m}) = \prod_i p_{\Delta t}(\Delta t_i \mid m_i) \tag{3.10}$$

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

$$P(\overline{m}) = \prod_i P(m_i) \tag{3.11}$$

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 data set (Rabiner and Juang, 1993). Finally, the most likely model sequence and segment sizes are

$$\overline{m}^*, \overline{\Delta t}^* = \max_{\overline{m}, \overline{\Delta t}} \prod_i p(\zeta_i \mid m_i, \Delta t_i) p_{\Delta t}(\Delta t_i \mid m_i) P(m_i) \tag{3.12}$$

or

$$\overline{m}^*, \overline{\Delta t}^* = \min_{\overline{m}, \overline{\Delta t}} \sum_i [-\ln p(\zeta_i \mid m_i, \Delta t_i) - \ln p_{\Delta t}(\Delta t_i \mid m_i) - \ln P(m_i)] \tag{3.13}$$

The above equation can be solved with dynamic programming (see for example (Cormen, Leiserson, Rivest and Stein, 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_{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 $p(\Delta t \mid m = m_{sil}) = U(140ms, 1s)$. A silence period shorter than 140ms is probably part of a word or a transition between words. It can therefore not be labelled "silence". We select $1s$ as the upper bound as there is no silence longer than that in our data set. $p(\Delta t \mid m = m_{sil})$ could also be trained instead of being preset but it would still be necessary to limit the shortest duration that a silence can be.

Table 3.1: A summary of the probability density functions (pdf) used in spike train classification. There are three spike train models for each word and there is one silence model.

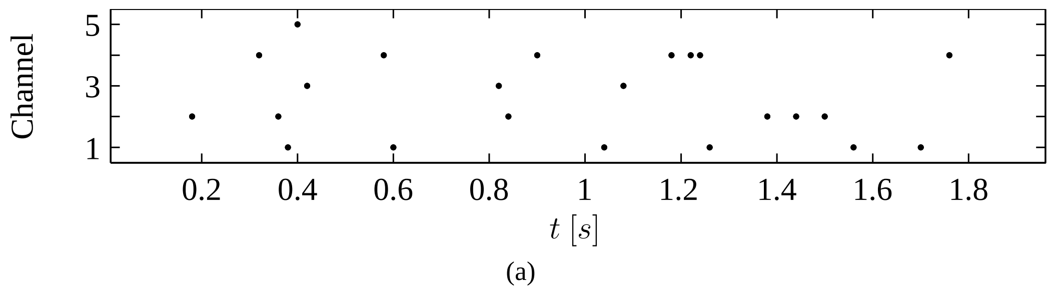
Pdf	Type	No. of components
$p(\zeta \mid m)$	Spike train model	34
For each channel in a spike train model we have:		
P_n	Discrete GMM	2
$f_t(t)$	Continuous GMM	3
p_a	Continuous GMM	2
Associated with each spike train model there is:		
$p_{\Delta t}$	Continuous GMM	2

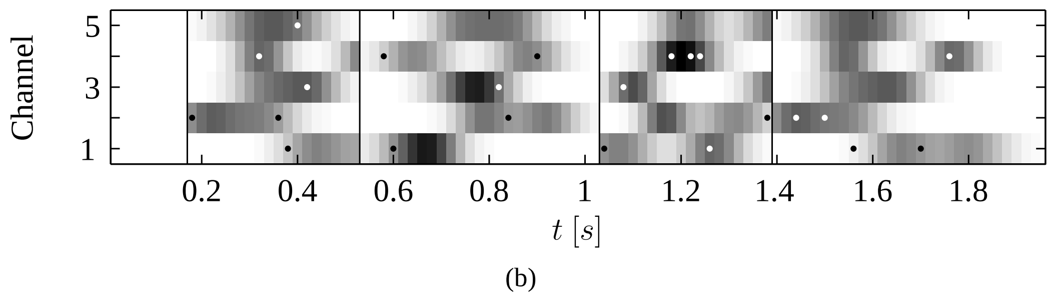
A set of three models is assigned to correspond to each word in the dictionary before training starts. We use more than one model per word because the spike trains for a given word may not all be similar. It is not easy to determine the optimal number of models per word class to use. Table 3.1 gives a summary of the probability density functions we use.

3.2.4 AN EXAMPLE OF SPIKE TRAIN CLASSIFICATION

In order to illustrate spike train segmentation and classification, we use the three models given in figure 3.2 as well as a silence model. The illustration is simplified by not taking the amplitude of spikes into account. Also for all three models $p(\Delta t \mid m)$ are set as uniform distributions, equal for all models; the prior probability of selecting the silence model and the prior probabilities of each of three other models are equal, so that $p(m) = 0.25$ with $m \in [1, 2, 3, m_{sil}]$.

Figure 3.3(a) shows a generated spike train that has to be classified. It is generated by sampling two spike trains from model $m = 1$, one spike train from $m = 2$ and one from $m = 3$. The spike trains are then concatenated in the order $m = [1, 3, 2, 1]$. The spike train is correctly classified; a period of silence is classified at the start of the spike train.





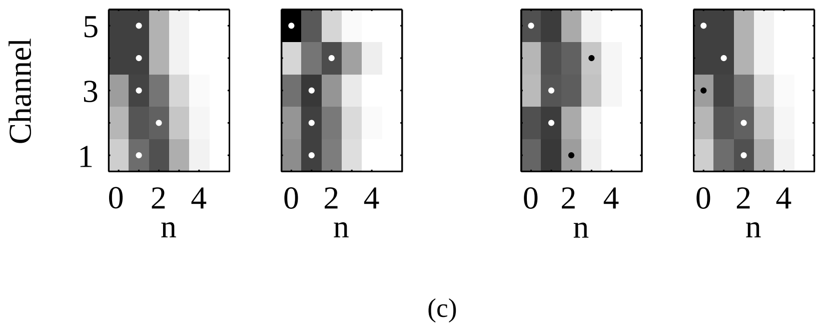


Figure 3.3: (a) shows a spike train that has to be segmented and classified. The spike train is correctly classified as $\overline{m} = [1, 3, 2, 1]$. The first part ($t = [1, 8]$) of the spike train is classified as "silence". (b) shows the segmentation as well as the normalized spike density function of each selected model. (c) gives the spike count distribution of each selected model.

3.3 TRAINING

In the training process the parameters of the spike train models are adapted to fit the data. We use expectation maximization (EM) to train the model. Firstly the expectation step finds the most likely sequence of models $\overline{m}^*$ and segment sizes $\overline{\Delta t}^*$ for the entire data set. The previous section on the classification of spike trains addressed exactly this issue. As the utterances are independent of each other, $\overline{m}_n^*$ and $\overline{\Delta t}_n^*$ can be determined for the nth utterance independently of all the other utterances in the data set.

The maximization step then adapts the model parameters to increase the likelihood of $\overline{m}^*$ and $\overline{\Delta t}^*$, or it maximizes the likelihood function $\mathcal{L}(\overline{\theta})$ with

$$\mathcal{L}(\overline{\theta}) = \prod_n p(\overline{m}_n^*, \overline{\Delta t}_n^* \mid \boldsymbol{\zeta}_n, \overline{\theta}) \tag{3.14}$$

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

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

$$\overline{\theta}^* = \min_{\overline{\theta}} - \sum_n \ln p(\overline{m}_n^*, \overline{\Delta t}_n^* \mid \boldsymbol{\zeta}_n, \overline{\theta}) \tag{3.15}$$

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.

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

3.3.1 UNSUPERVISED TRAINING

The data sets that are used for the training of speech recognition systems may be labelled. Labeled data sets are usually expensive to gather because each utterance has to be labelled. Unlabeled data sets on the other hand can very easily be gathered by recording speech from radio, television, phone conversations etc.

When models are trained on unlabeled data or when the label information is not used, the training procedure is unsupervised. Spike train models can be trained in an unsupervised fashion. There is one conceptual difficulty when using unsupervised training. Each EM iteration changes the data set because it changes $\overline{\Delta t}^*$. If EM converges to the global minimum, the fact that the data set changes each iteration would not influence the final solution. However, EM only finds a local minimum that is dependent on the data set, and the changing data set means that the minimum may not be a good enough solution to get the desired performance from the model.

We use supervised training so that we circumvent the problems of training with EM, and it also allows us to force spike train models to correspond to words. However unsupervised training of spike train models has one big advantage over supervised training: it would show which speech units are statistically significant. It would be an important result if the speech units that spike trains code compare well with speech units measured in the auditory cortex. A spike train classifier trained in an unsupervised manner may perform better than one trained in a supervised manner as it is free to choose its the speech units.

3.3.2 SUPERVISED TRAINING

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 may be that the spike trains of a certain word class are grouped into several clusters of similar spike trains. If this is the case, then the classification performance will depend on the number of clusters that exist, and also whether these clusters are discovered during training. It is not easy to determine the optimal number of models per word class to use.

The expectation step uses a modified version of the standard dynamic programming algorithm used during the Viterbi search (Ostendorf, Digilakis and Kimball, 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 figure 3.4). 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. Figure 3.5 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

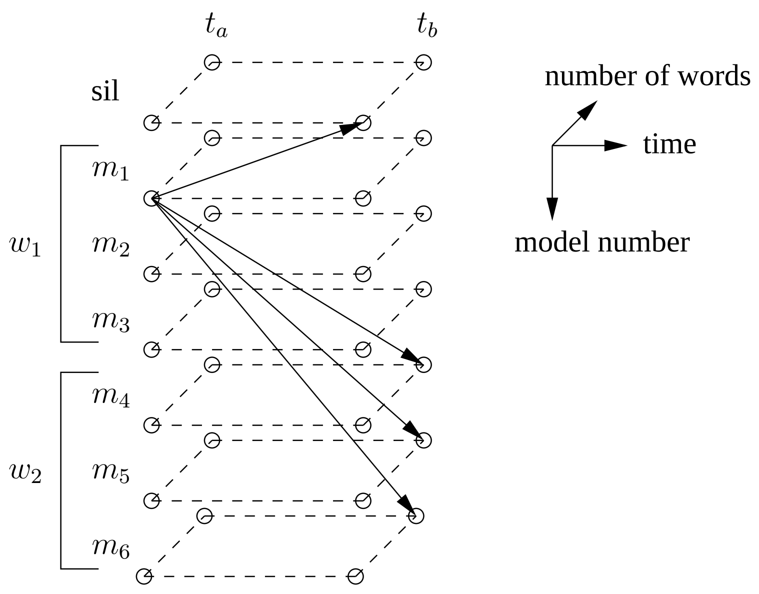


Figure 3.4: From a vertex there are only two types of valid transitions to a given time instant. There is the transition to the silence model, for which the number of words since the start of the sequence do not increase; and there is the transition to all the models in the set that correspond to the next word in the sequence.

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_{sil}.

3.3.3 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 $\bar{\theta}_{start}$ to determine the most likely model sequence $\overline{m}^*$ and segment sizes $\overline{\Delta t}^*$. It then adapts the model parameters to better fit $\overline{m}^*$ and $\overline{\Delta t}^*$. We do not know what a good starting point should be, so we skip the first step

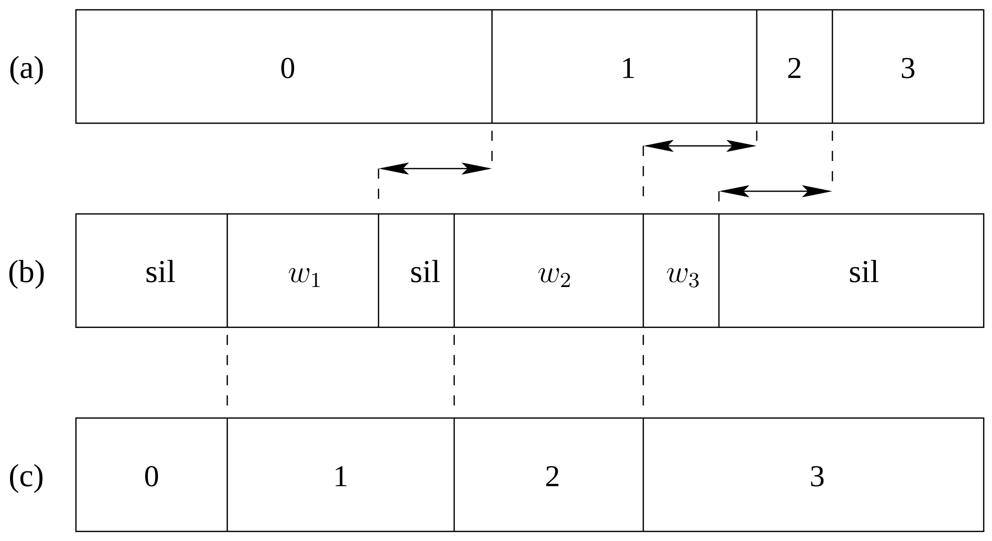


Figure 3.5: This figure shows how the number of words since the start of the sequence is bounded. (b) The target sequence of words. (a) The minimum number of words that has to be in the sequence at a specific time. This is determined by the word-end boundaries. A word has to occur before the time of 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 $380ms$). (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.

of the first iteration of EM and manually set $\overline{m}^*$ and $\overline{\Delta t}^*.\overline{\Delta t}^*$ is set according to the data set labels. $\overline{m}^*$ is set by dividing all the samples in a word class evenly between the models of that class. For example, the first occurrence of "three" in the data set is labelled as $m = 7$, the second 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.

3.4 IMPLEMENTATION DETAILS

There are two important details we implement for the classification of spike trains. Firstly we force the spike train models to correspond to words in order to keep the classification simple. Secondly we process the spike train before it is classified.

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 (see figure 3.6) against different threshold values and two values of λ. We use $\lambda = 0.3$ and choose the threshold as 0.124 so that only 40% of the spikes remain. 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.

This step resembles *sparse code shrinkage* (SCS) (Hyvärinen, 1999b). SCS is a post processing step that applies to sparse codes calculated with ICA. ICA does not explicitly model noise, the code elements therefore also reconstruct the noise. Very small code values probably reconstruct noise but the shrinkage function removes these elements. A typical shrinkage function is

$$h(a) = \text{sign}(a) \max\left(0, |a| - \sqrt{2}\sigma^2\right) \tag{3.16}$$

where σ^2 is the variance of noise. Other shrinkage function can be derived based on assumptions about the noise and the prior probability $p(a)$. SCS is a rigorous method to denoise an ICA code.

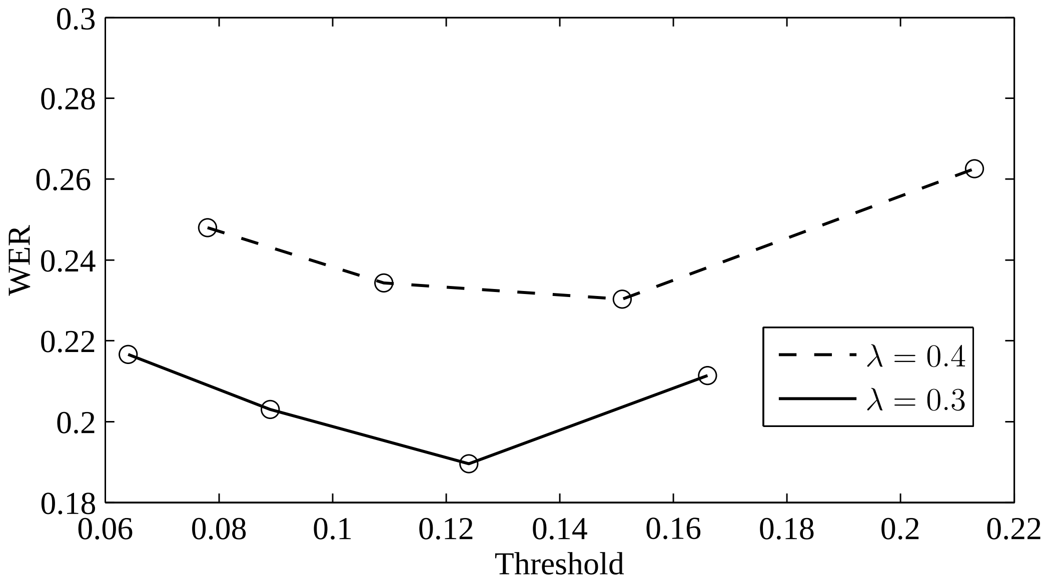


Figure 3.6: The figure shows the classification performance (Word Error Rate) of the system as a function of the threshold. Spikes with an amplitude less than the threshold are removed from the code. Each circle on a line corresponds to certain percentage of spikes that remain in the code. From the left most circle to the right the percentages are for each line: 60%, 50%, 40% and 30% respectively. The two lines are for different values of λ.

The sparse coding framework we use model noise explicitly, it is contained in $\lambda = 2\sigma^2$. Although our post processing step and the shrinkage function of SCS appear to be similar, they serve a different purpose. Our post processing step is performed because the sparseness function is not ideal, whereas the shrinkage function removes noise.

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 order in 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 $\mathbf{\Phi} = \left[\mathbf{\Phi}_{\{0\}}, \mathbf{\Phi}_{\{1\}}, \mathbf{\Phi}_{\{2\}}, \ldots, \mathbf{\Phi}_{\{6\}}\right]$. We can split the code $\boldsymbol{a}$ in the same manner; $\boldsymbol{a} = \left[\boldsymbol{a}_{\{0\}}, \boldsymbol{a}_{\{1\}}, \boldsymbol{a}_{\{2\}}, \ldots, \boldsymbol{a}_{\{6\}}\right]$ where $\boldsymbol{a}_{\{s\}}$ refers to the sparse code associated with $\mathbf{\Phi}_{\{s\}}$. The compressed code is then

$$\hat{\boldsymbol{a}} = \sum_{s=0}^{6} \boldsymbol{a}_{\{s\}} \tag{3.17}$$

3.5 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 3.2 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 confusible initial phonemes.

Figure 3.7 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

Table 3.2: The confusion matrix of classification of the test set. The diagonal entries are words that are correctly classified; the off-diagonal entries are words that have been replaced by incorrect words during classification.

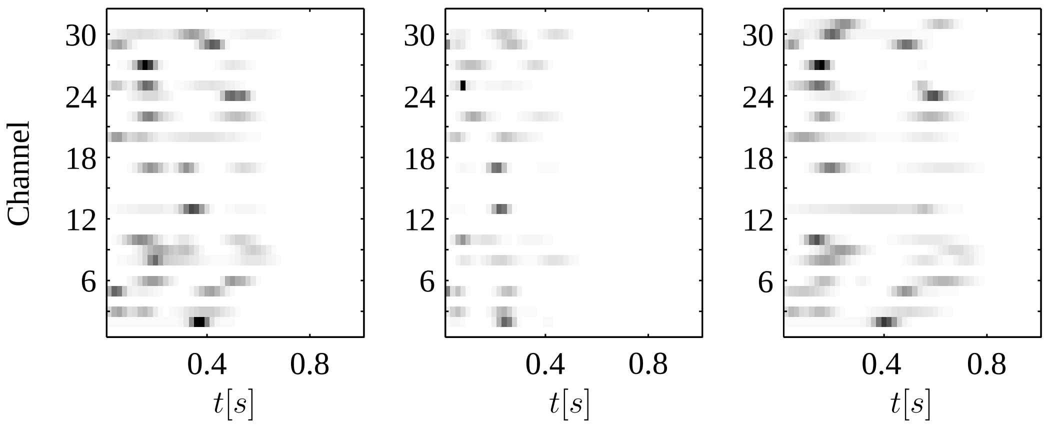
		True class										
		one	two	three	four	five	six	seven	eight	nine	zero	oh
Recognized class	one	240	2	2	14	4	0	0	9	12	1	8
	two	0	266	5	2	0	1	5	7	3	3	1
	three	2	9	283	1	0	0	0	13	0	1	0
	four	7	4	1	260	5	1	0	0	0	0	3
	five	2	3	2	11	230	1	3	1	8	2	19
	six	1	3	1	0	1	288	2	0	0	0	0
	seven	0	10	2	2	1	8	259	1	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

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.

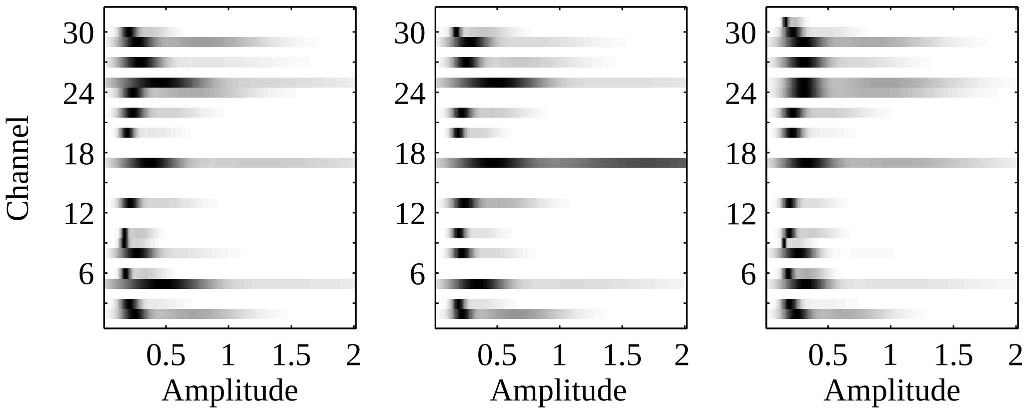
Figure 3.8 shows how often each model is used ($P(m)$) as well as the lengths of spike trains that are coded by each model ($p_{\Delta t}(\Delta t \mid m)$). We see from $p(\Delta t \mid 16)$ and $p(\Delta t \mid 18)$ that the models will mostly be selected when the time span Δt is rather long. The time 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, $p_{\Delta t}(\Delta t \mid m)$ 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_{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 figure 3.8 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 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.

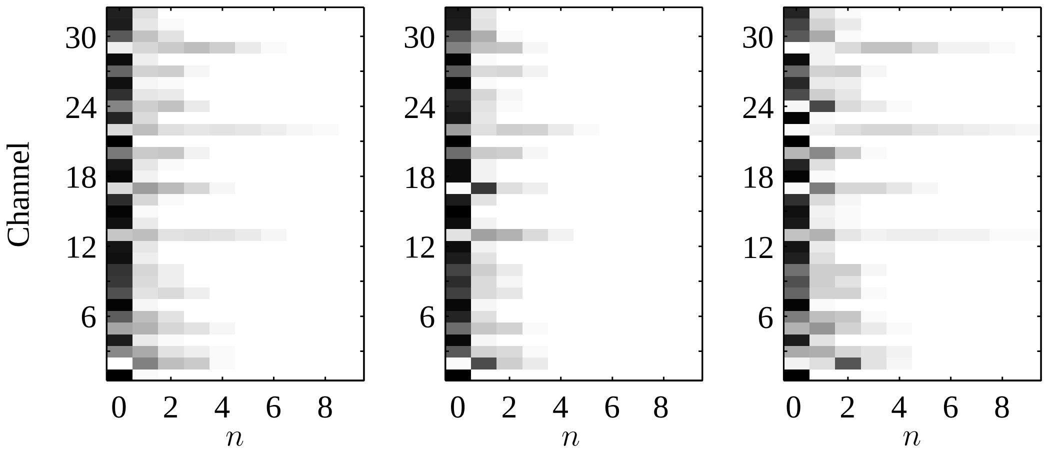
The HMM system achieved a WER of 15% for models based on monophones. The models were single mixture components with full covariance matrices. Straightforward HMM-based speech recognition with triphones achieves a 3% WER on the TIDIGITS dataset when Mel Frequency Cepstral Coefficients (MFCCs) are used 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 in the next chapter.



(a) p_t



(b) p_a



(c) P_n

Figure 3.7: 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 $P_n(0) \leq 0.2$

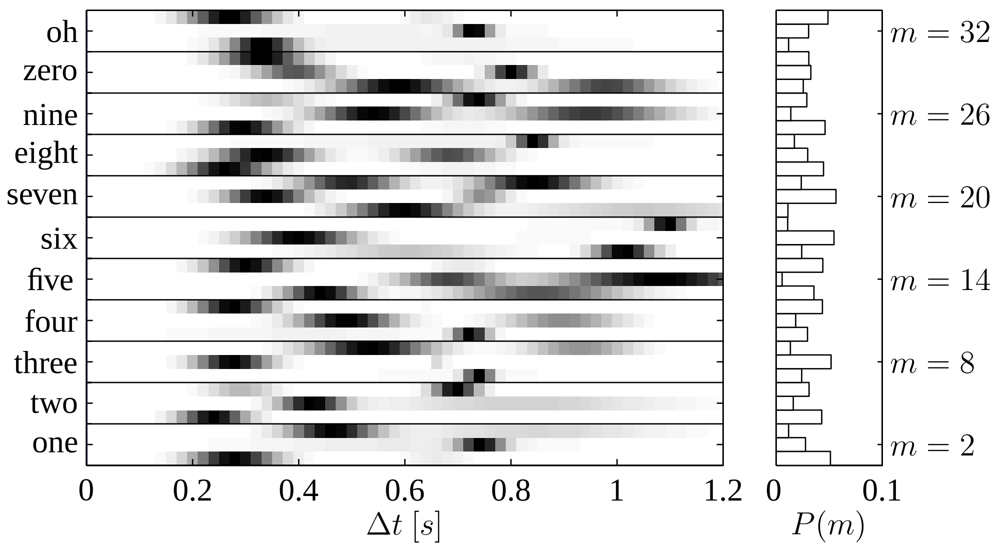


Figure 3.8: The *left* figure gives $p_{\Delta t}(\Delta t \mid m)$; the *right* figure gives $P(m)$. There are 33 rows in $p_{\Delta t}(\Delta t \mid m)$ and in $P(m)$, each row corresponds to a model. For example $p_{\Delta t}(\Delta t \mid 1)$ and $P(1)$ appear in the bottom row of each plot while models $m = 7$ to $m = 9$ are associated with the word “three”.

Chapter Four

Discussion

We have shown how sparse coding can be used for speech recognition. The steps to classify an unknown speech waveform are as follows:

1. The raw speech waveform is transformed in a frequency domain representation,
2. A sparse code that represents the signal is found by minimizing a cost function. The cost function consists of two terms: a reconstruction error and a sparseness function. The code that gives a low reconstruction error but that is also sparse is the code we select to represent a signal.
3. The sparse code is classified by searching for the most likely sequence of spike train models that fit the sparse code. Each spike train model is associated with a word from the dataset, thereby giving us the classification of the sparse code.

We found that for a relative simple signal representation and simple dataset, the system has a WER of 19%. This compares well with the performance of a HMM-based system which achieves a WER of 15% on the same dataset using the same coarse signal representation.

4.1 SPARSE CODING

4.1.1 THE DICTIONARY

Neuroscientists would like to understand the neural code better. Research are being conducted on several fronts, sparse coding being one of them. Thus far sparse coding is successful at explaining the receptive fields of some neurons. Dictionaries for image patches compare well with the receptive fields of simple cells in the visual cortex. It would be very interesting to see whether dictionaries for spectrograms are also related to properties of cells in the auditory pathway. However the dataset should then be natural sounds and not spoken digits.

The trained dictionary has some elements that are on syllable level. This is encouraging as the basic unit of speech is also believed to be on that level. It shows that there are at least some agreement between the dictionary and the receptive fields of neurons in the auditory pathway. It is necessary to investigate whether this property holds for more complex datasets. The digit dataset we use has very short words, most of which are single syllable words. We cannot conclusively say that sparse coding with a LGM yields a dictionary on the syllable level.

Sparse coding with a LGM can only explain the receptive fields of a quarter of cells in V1. The remaining cells have more complex receptive fields. The activity of complex cells is also sparse, but the LGM is not complex enough to model these cells. We showed in figure 2.17 that the activity of code elements is not independent as we would like them to be. This demonstrates that the LGM is limited in the order of statistical structures it can capture. A more complex model could produce receptive fields of complex cells and thereby giving a more powerful sparse code.

The size of the dictionary plays an important role in the properties of the dictionary elements and of the code. The code will be maximally sparse if the dictionary has as many elements as there are basic features in the dataset. Normally we do not know how many features there are in a dataset, this makes it difficult to determine the optimal dictionary size. If the dictionary size is increased while λ 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. There will be very few samples where each dictionary element is used, which in turn will reduce the classification performance of the system.

We did not study the effect the size of the dictionary has on the type of features that the dictionary elements code or on the classification performance. Here more work needs to be done.

The dictionary training is stopped prematurely because the training takes a long time. There is a small improvement in the error function from one iteration to the next when it is stopped. Further training will reduce the error function more which should improve the classification performance. This too needs to be investigated.

The elements are linearly scaled to account for the fact that some sounds may be streched in time without changing the meaning of the sound. Phonemes however do not scale linearly. The classification performance may improve if the scaling mechanism better resembles the nonlinear scaling of actual phonemes. A more complex scaling mechanism, such as dynamic time warping, would increase the computational cost of finding a sparse code even more. It is necessary to study the effect of some nonlinear scaling mechanisms.

4.1.2 THE BALANCE BETWEEN RECONSTRUCTION ERROR AND SPARSENESS

We need a method to tell us how big a dictionary should be given the dataset and the value of λ. The number of phonemes and syllables that occur in a dataset should play an important role in determining the size of the dictionary. It is reasonable to expect that a dictionary which codes spoken digits will have to be smaller than a dictionary which codes conversational

English. The value of λ determines how well a signal will be reconstructed. If λ has a low value, fine detail in the signal is reconstructed. In this case the elements of a too small dictionary will be overused, that is they will have to be used in coding almost every signal, which will not allow them to capture the underlying structure.

The effect of λ on the performance also 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 $\lambda < 0.3$. There should be a turning point for which a reduction in λ decrease the classification performance.

4.1.3 CONSTRAINING THE DICTIONARY NORMS

The fact that we chose to constrain the norms of dictionary elements plays an important role in the features that the dictionary code. Dictionary elements have to be constrained because in the problem formulation the integral in $p(\bar{x}) = \int p(\bar{x} \mid \bar{a})p(\bar{a})d\bar{a}$ (equation 2.15) is estimated by a delta function. When the integral can be more accurately evaluated the dictionary norms will not have to be constrained. The dictionary elements will then adapt in such a way that the code elements follow the prescribed prior probability $p(\bar{a})$. We have a good idea of what the prior probability should be for a true neural code where the code elements are binary. It can be estimated from the average firing rate of the neuron. However it is not clear what the prior probability for real valued code elements should be. It may even be different for different neurons.

The problem of constraining the norms of dictionary elements is not unique to gradient based algorithms. Codes that are determined with basis selection methods also suffer from this problem. MP has to have some mechanism that controls the norms of dictionary elements so that they stay competitive (Perrinet, 2004b). x

4.1.4 FINDING A SPARSE CODE

The problem of finding a sparse code is not yet satisfactorily solved. We use an iterative optimization approach that is computationally expensive. Although this algorithm is substantially faster than other gradient based algorithms, there is still a need for an even faster algorithm. In order to find sparse codes in reasonable time, we use a sparseness function that deviates significantly from the ideal sparseness function. Unfortunately with our sparseness function the sparse code has many spikes with very small amplitudes. We found that by removing 60% of the spikes the performance increases significantly. The spikes that are finally removed put unnecessary strain on the algorithm while it searches for a sparse code. These spike are included in every iteration even though they do not contribute to the solution. The algorithm will execute much faster if there are 60% less spikes in the code.

It will be better if the sparseness function has a form that not only includes the code element amplitudes but also the activity of code elements. This sparseness function will produce a code that does not require the removal of many spikes to give good performance. Such sparseness functions exist, but they cause the optimization problem to become non-quadratic and computationally much more expensive. Future work should focus on how to solve sparse codes efficiently, especially sparse codes of sparseness functions that include the activity of code elements.

We used a final optimization step for MP which we termed MP+. This step simply changes the values of the nonzero code elements to minimize the reconstruction error. It is a step that can reduce the reconstruction error of any code without increasing its l_0 norm. The classification performance of the system may improve if this step is also performed on the SSQP codes. We did not do any investigations along this line.

Basis selection algorithms such as MP are generally computationally cheap. They also use the l_0 sparseness function which better fits the neural code than the sparseness function used by gradient based algorithms. Be that as it may we found that MP does not give robust codes. A small change in the signal leads to a big change in the code. Other basis selection algorithms should be evaluated based on the requirement of robust codes and computational efficiency.

The quadratic sparseness function S_{quad} is a much better function to use than the simpler l_1 norm. S_{quad} codes have fewer nonzero code entries than l_1 norm codes for a given reconstruction error.

4.2 SPIKE TRAIN CLASSIFICATION

4.2.1 THE SPIKE TRAIN MODEL

The spike train models are able to decode a spike train successfully. 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. At some stage it is necessary to use supervised training of the recognition part of the system. It may merely be to determine the sounds that each model codes.

The spike train model actually finds higher-order correlations among code elements. It uses these correlations to decode the spike train. A coding model that is more complex than the LGM may be able to model these higher-order correlations. If this is the case, then the spike train becomes very simple to decode; the activity of certain neurons may signal when a certain word is present in the utterance. A spike train can then still be decoded with the spike train model, but the model will be much simpler, and the result may be more accurate.

The classification of spike trains is done on trains that do not include the scaling of dictionary elements in any way. We showed earlier that 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.

4.2.2 EXPECTATION MAXIMIZATION

The EM algorithm finds a solution for the spike train models that is a local minimum. For example the results show that two of the three models that correspond to the word “six” are very similar and both have unreasonable long time spans. These models could be used better, but the EM algorithm is stuck. The classification performance can potentially increase if the EM algorithm is guided more in its search for a minimum. The guidance can be user input or some automatic process.

The problem that EM gets stuck in local minima is a property of the algorithm and not of the dataset. Any improvements of the EM algorithm will also be useful here.

4.3 GENERAL REMARKS

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.

We did not check the performance of the system with noise maskers. There is good reason to believe that sparse coding will perform well on noisy signals, as sparse coding can also be used as a means to remove noise from a signal (Hyvarinen, 1999a; Shang, Huang, Zheng and Sun, 2006). In other words, sparse coding captures the underlying structure of the signal and thereby is not so susceptible to noise.

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 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.

It will be interesting to use the same approach for image recognition as we use here for speech recognition. We have a more intuitive understanding of vision than we have of speech. It will be easier to interpret the dictionary elements as well as the spike train models.

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

4.4 SUMMARY

This thesis shows that sparse coding can be used to do continuous speech recognition. It highlights some factors that currently limit the use of sparse coding for pattern recognition. One such factor is the computational effort required to find a sparse code. We propose the SSQP algorithm which it is much faster than popular gradient based algorithms. The thesis further showed that useful spectro-temporal features can be extracted from speech when the LGM dictionary is trained. This is an important point as most other speech recognition systems use across-frequency features. We have also showed that the spike train model with order statistics can successfully decode spike trains of variable length. This initial investigation emphasizes the areas that need further study before sparse coding will be more widely used.