AUTOMATIC RECOGNITION OF BIRDSONG SYLLABLES USING
ULTRA-SPARSE SPIKE SEQUENCE REPRESENTATIONS

A Dissertation in
Physics
by
Leonardo Estêvão Schendes Tavares

© 2022 Leonardo Estêvão Schendes Tavares

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

December 2022
The dissertation of Leonardo Estêvão Schendes Tavares was reviewed and approved by the following:

Dezhe Z. Jin
Associate Professor of Physics
Dissertation Advisor
Chair of Committee

John C. Collins
Distinguished Professor Emeritus of Physics

Chad Hanna
Professor of Physics & Astronomy and Astrophysics

Vijaykrishnan Narayanan
Robert Noll Chair Professor of Computer Science & Engineering and Electrical Engineering

Nitin Samarth
Professor of Physics
George A. and Margaret M. Downsbrough Department Head
Abstract

Birdsong shares many features with spoken language while remaining free from coarticulatory effects. Birdsong syllables are stereotyped, acoustically complex, and connect by silent gaps within motifs and phrases. Songbird species show different forms of complexities in their songs. In the zebra finch, for example, while motifs are stereotyped and have a fixed syntax, syllables are harmonically rich. Canaries use complex syntactic rules which can span the duration of many phrases, but, acoustically, their syllables are much simpler. These differences and simplifications make birdsongs useful test beds for developing new approaches for continuous recognition of vocal units. We first introduce a recognition system for Bengalese finch songs that uses ultra-sparse sequences of spikes to represent syllables. We then develop a reconstruction method for recognizing syllables in continuous recordings. Incidentally, we find that this system recognizes Bengalese finch syllables at low error rates while using only a few seconds worth of syllables for training. We then develop a series of modifications in an attempt to extend the system for canaries. In particular, we introduce an onset/offset representation of syllables to deal with the problem of temporal warping. We find that the level of similarity and acoustic variation amongst some syllable types make it difficult to reduce the error rates. To deal with this issue, we introduce the concept of binary detectors. Binary detectors deal with the problem of compositionality in the vocal repertoire when the lexicon is unknown. We show how ensembles of boosted binary detectors can help during recognition of canary syllables. Comparing our results with the literature, we find that our system achieves the state-of-the-art recognition accuracy with about a 10-fold reduction in total duration of training syllables. Our results emphasize the important role of using ideas from neuroscience in guiding research in machine intelligence. In the last chapter, we switch to a different topic to introduce a method for reconstruction of neuronal somata from slice imaging. We show how the surface area and the volume of somata can be estimated and compare our reconstructions of neurons in the mouse V1 to popular databases.
# Table of Contents

List of Figures vii

List of Tables xi

Acknowledgments xii

## Chapter 1
Introduction 1

1.1 Birdsongs .................................................. 2
1.1.1 Birdsong production and acoustics ......................... 2
1.1.2 Automatic recognition of birdsong syllables ............... 5
1.2 Machine hearing ............................................. 7
1.2.1 Feature processing ....................................... 7
1.2.1.1 Gammatone filters .................................... 8
1.2.1.2 Midlevel processing ................................. 10
1.3 Overview .................................................. 12

## Chapter 2
Speech recognition with spike sequences 14

2.1 Spike-based representation of speech .......................... 14
2.1.1 The STRF ................................................. 15
2.1.2 Feature detectors ....................................... 16
2.1.3 Sequences of spiking detectors ............................ 17
2.2 Speech recognition using spike sequences ...................... 17
2.2.1 Isolated recognition with spike sequences ................. 18
2.2.2 Towards continuous recognition with spike sequences ... 20
2.2.2.1 Segmentation using rapid spectral changes ............ 20
2.2.2.2 Recognition using relative delays .................... 24
2.3 Conclusions ............................................... 26

## Chapter 3
Continuous recognition of vocalizations of fixed-duration 27

3.1 Syllables of fixed-duration .................................. 28
3.2 Spike-based representation of syllables ....................... 30
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.1</td>
<td>Syllables as ultra-sparse spike sequences</td>
<td>31</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Training</td>
<td>35</td>
</tr>
<tr>
<td>3.2.2.1</td>
<td>Selection of acoustic events</td>
<td>35</td>
</tr>
<tr>
<td>3.2.2.2</td>
<td>Template training</td>
<td>39</td>
</tr>
<tr>
<td>3.3</td>
<td>Continuous recognition of ultra-sparse sequences</td>
<td>40</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Sequence detection with delay lines</td>
<td>41</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Formation of auditory objects using a simple delay grid</td>
<td>43</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Balancing insertions and deletions</td>
<td>44</td>
</tr>
<tr>
<td>3.3.4</td>
<td>Sequence matching using delay distributions</td>
<td>45</td>
</tr>
<tr>
<td>3.3.5</td>
<td>Search-based sequence reconstruction</td>
<td>50</td>
</tr>
<tr>
<td>3.3.5.1</td>
<td>Spike-alignment graph</td>
<td>50</td>
</tr>
<tr>
<td>3.3.5.2</td>
<td>Sequence completion</td>
<td>53</td>
</tr>
<tr>
<td>3.3.5.3</td>
<td>The prediction stream</td>
<td>55</td>
</tr>
<tr>
<td>3.3.6</td>
<td>Feature manipulation</td>
<td>57</td>
</tr>
<tr>
<td>3.3.6.1</td>
<td>Spectral entropy</td>
<td>57</td>
</tr>
<tr>
<td>3.3.6.2</td>
<td>Feature augmentation</td>
<td>58</td>
</tr>
<tr>
<td>3.3.7</td>
<td>Continuous recognition of Bengalese finch syllables</td>
<td>60</td>
</tr>
<tr>
<td>3.3.7.1</td>
<td>Limitations</td>
<td>63</td>
</tr>
<tr>
<td>3.4</td>
<td>Conclusions</td>
<td>63</td>
</tr>
</tbody>
</table>

Chapter 4

Continuous recognition of vocalizations of variable duration 65

4.1 Acoustic variability in canary syllables 66

4.2 The on/off representation 67

4.2.1 Syllables as on/off sequences 68

4.2.2 Reading the on/off representation 72

4.3 Search-based recognition 72

4.3.1 Spike-alignment graph revisited 72

4.3.2 Phrase recognition 75

4.3.3 Syllable recognition 77

4.3.3.1 Dealing with acoustic variations 77

4.3.3.2 The prediction stream revisited 79

4.3.3.3 Bengalese finch and canary syllable recognition 80

4.4 Continuous recognition with binary detectors 83

4.4.1 Binary feature detectors 83

4.4.2 Boosted binary detectors 84

4.4.3 Binary ensembles of spectrotemporal patches 85

4.4.4 Syllable recognition with boosted binary detectors 88

4.5 Conclusions 90

Chapter 5

Continuous recognition and compositionality in birdsongs 92

5.1 Recognition in the presence of shared subunits 93

5.1.1 Rule-based decisions 93
List of Figures

1.1 Birdsongs and speech ......................................................... 5
1.2 Gammatone filterbanks ...................................................... 10
1.3 Midlevel processing for continuous recognition of syllables .......... 11

2.1 Recognition of isolated spoken digits under noisy conditions using spike
sequences ................................................................. 19
2.2 Feature detectors for CV transitions ....................................... 21
2.3 Feature detectors for groups of CV transitions ............................. 23
2.4 Segmentation using transition detectors .................................... 24
2.5 Using relative spike delays for isolated recognition ...................... 25

3.1 Temporal warping in a Bengalese finch syllable .......................... 28
3.2 Temporal warping in a canary syllable .................................... 29
3.3 Duration and distortion distributions of Bengalese finch and canary syllables 29
3.4 Encoding short-time acoustic events in birdsong syllables .............. 32
3.5 Encoding birdsong syllables as sequences of short-time events .......... 34
3.6 Syllables as sparse templates .............................................. 35
3.7 Clustering short-time events in syllables .................................. 36
3.8 Preventing complex decision boundaries when syllables share features .. 38
3.9 Relative delays in templates of syllables ........................................ 40
3.10 Delay grid of LIF neurons for temporal pattern matching ................. 42
3.11 Delay grid of feature detectors for template matching ....................... 43
3.12 Generalization error using delay lines for recognition ..................... 46
3.13 The effects of insertions and deletions in the generalization error ........ 47
3.14 Generalization error in the presence of background sources ............. 47
3.15 Distributions of relative delay .................................................. 49
3.16 Spike alignment nodes and graph .............................................. 51
3.17 Weighted spike alignment graphs .............................................. 53
3.18 Sequence completion with an alignment node ................................ 54
3.19 Multi-resolution spectral entropy .............................................. 58
3.20 Feature augmentation with frequency translation ............................ 60
3.21 Benchmarks in a recognition system for Bengalese finch syllables ....... 62

4.1 Example of acoustic variability in canary syllables .......................... 67
4.2 Contrasting spike representations for time-varying syllables ............... 68
4.3 Onsets/offsets in SVMs. .................................................... 69
4.4 Onset and offset syllable templates ........................................... 70
4.5 Onsets/offsets and temporal warping in canary syllables. .................. 71
4.6 Ambiguities and issues in the alignment graph ................................ 74
4.7 Recognition of phrases in the canary ......................................... 76
4.8 Feature detectors and acoustic variability ..................................... 78
4.9 Anchor points for syllable segmentation ...................................... 80
4.10 Benchmarks for Bengalese finch syllables......................... 81
4.11 Benchmarks for Bengalese finch syllables in the absence of noise .... 82
4.12 Benchmark for canary syllables .................................. 82
4.13 Binary feature detectors ........................................... 84
4.14 Boosted binary detectors ........................................... 87
4.15 Training and empirical errors of boosted binary ensembles .......... 89
4.16 Benchmarks for Bengalese finches using binary detectors ............ 89
4.17 Benchmarks for a canary using binary detectors ..................... 90
4.18 Benchmarks for a different lineage of Bengalese finches .......... .... 90

5.1 Detecting note transitions with the prediction stream ............... 96
5.2 Decision rules with binary detectors ................................ 98
5.3 Decoding with SVM-based binary detectors ......................... 100
5.4 Two-layer decision system for continuous recognition of syllables .. 101
5.5 Gaussian kernels for detection with onsets and offsets ............... 103
5.6 Delay line does not support time-varying patterns of onsets/offsets ... 105
5.7 Clustering onsets and offsets implements pooling .................... 106
5.8 Spectral clustering for on/off patterns ................................ 107
5.9 Pooling for note composition in the song sparrow ................. ... 108

6.1 Measuring somatic area ............................................. 111
6.2 Soma reconstruction from image stacks ................................ 113
6.3 Reconstruction of the surface area of known shapes ................. 114
List of Tables

3.1 Summary of Bengalese finch songs used for the benchmarks in Fig. 3.21. 63
Acknowledgments

This dissertation is the culmination of the support of many. I thank my advisor, Dezhe Jin, for his mentorship and support. I have learned a great deal about research and creativity from Dezhe, and I am eternally grateful to him. My colleagues in the Jin Group, who have helped me more times than I could count, Sumithra Surendralal, Eugene Tupikov, Jiali Lu, Autumn Zender, Kevin Sargent, Aayush Khare, and Derek Sederman. Our collaborators at OHSU, Henrique von Gersdorff, Benjamin Zemel, Alex Nevue, and Claudio Mello for the exciting opportunity to learn more about songbirds. Jon Sakata, for kindly sharing his Lab’s data with us. I also thank the members of my committee, John Collins, Chad Hanna, and Vijay Narayan. I am grateful to the faculty at the Department of Physics, particularly Dan Costantino, Kirstin Purdy Drew, and Stephen Van Hook, for the opportunity to teach so many physics classes over the years, and Rick Robinett for the support. To the physics staff, including Carol Deering and Juli Mortimore for all their help. I am in debt to all my many friends at Penn State I had the luck to meet throughout these years. They made it all worth the while. Sumithra, Ssohrab, Umar, Sean, Eugene, Jiabin, Monica, Jiali, Yang, Neel, Patrick, Nithin, Lakshmy, Egon, Jiho, Krishna, Autumn, Ya-Wen, Tsung-Yao, Kevin, Francisco, Emma, Peter, Tommy, Emmy, Lisa, Justin, Dan, Tim, Ben. Nora and Lori, who helped me so much. Marty and Debbie for sheltering me. My eternal friends from SP, Bruno, Pedro, Olcyr, Dobay, Nath, Ricardo, Bill, all my many friends from tango. My friends from BH, who I also miss. My family, for the endless and unconditional support. Mãe, Pai, Vovó. My partner Laura, without whom I would be lost.

This work was partially supported by the NSF award EF-1822476 and the Huck Institutes of the Life Sciences at The Pennsylvania State University. The findings and conclusions do not necessarily reflect the view of the funding agency.
Chapter 1
Introduction

Although automatic speech recognition (ASR) technology has improved in the last decade, it is still not clear how it will scale in the future. Bottlenecks which limit the use of ASR in practice include energy consumption, memory usage, storage space, training data, and computational power. Meanwhile, animals have seemingly devised very efficient and robust ways to deal with vocal communication. This includes not only speech, but also birdsongs [1]. Although the mechanisms behind it remain poorly understood, it’s been hypothesized that sparse spike representations may be an important working principle. However, understanding how to construct spike-based systems for speech-to-text or related technology has remained a challenging problem.

Like speech, birdsong is a complex learned vocal behavior. But birdsong is usually much simpler. Because it is a courtship behavior, where precise mimicry is usually selected for, birdsong is much less variable than speech. Songbirds are also unable to produce consonant-like sounds. Additionally, their breathing patterns during singing introduce short silent pauses between syllables. These and other simplifications make birdsongs interesting for testing new ideas in automatic recognition.

In this dissertation, we develop a spike-based recognition system for continuous automatic labeling of birdsong recordings. In the following sections of this introductory chapter, we review some background concepts. First we discuss birdsong production and automatic recognition. Then we describe the front-end of our recognition system. At the end of the chapter, in Sec. 1.3, we outline the organization of the dissertation. Here we summarize our most significant contributions and their respective chapters:

- Ch. 3: Non-linear spiking feature detectors for spectrotemporal patterns, template matching of spike patterns using delay grids and graph search, and dynamic thresholds for spike sequence completion;
• Ch. 4: Onset/offset representation for vocalizations of variable duration and binary feature detectors for feature compositionality;

• Ch. 5: Binary detectors for note compositionality and pooling for temporal warping and song segmentation.

We have made our code, including a graphical interface for song annotation, available and open source at https://github.com/leoschendes/BASS.

1.1 Birdsongs

Songbirds produce a great variety of acoustic signals, ranging from tonal sounds to low-pitch and harmonically rich calls. Birdsongs are distinct in that they are learned complex behaviors. Like speech, songs can be broken down into smaller constituents. The smallest unit is the note. Notes play the analogous role of phones in speech. On the next level, in parallel with words, are syllables. Syllables can be formed by two or more notes glued together or spaced by a very short silent gap. Frequently, however, the syllable is a single entity. Syllables are typically sung in sequence to form motifs or phrases. Here we find a major difference between speech and birdsongs: coarticulation. While words in spoken language tend to merge one into another, thus altering the acoustical properties of phones, syllables are always preceded and followed by silent gaps. The duration of this gap can sometimes be short and fixed for some species (for example in the zebra finch song, bottom panel of first column of Fig. 1.1) or large and variable in others (see the canary phrase on the top of the second column of Fig. 1.1). In fact, song complexity varies considerably from species to species. For example, in the Bengalese finch, the duration of syllables can be extremely rigid. As we will explore in this dissertation, these properties make birdsongs great test beds for developing new technologies for speech processing. In this section, we briefly describe the acoustical mechanisms of birdsongs production. We then review previous work done on the problem of birdsong recognition.

1.1.1 Birdsong production and acoustics

Different from mammals, which use their larynx for phonation, birds use a distinct organ called the syrinx to vocalize. The anatomy of the syrinx varies from species to species, but in most oscines the membranes responsible for phonation are located near the interface between the trachea and the bronchi. These are two pairs of connective tissue masses called the lateral and medial labium (LL and ML, respectively), each pair
in one bronchus. Muscles in the vicinity adjust LM and MM to regulate amplitude and frequency. Syringeal and respiratory muscles also mutually coordinate to allow the bird to take short bursts of breath between each syllable. This results in songs composed of short syllables, each surrounded by a small silent gap. In contrast, humans typically take a long breath before vocalizing entire utterances. Thus, individual words in casual speech are commonly merged one into the other without noticeable silent gaps.

Although songbirds lack vocal folds, their mechanism of phonation is in principle similar to that in speech [2, 3]. First the bird moves the syrinx to a phonatory position. This step consists of contracting and extending a set of muscles at the periphery of the two bronchi in order to partially constrict them. At the tip of each constriction lie one pair of syringeal membranes. Airflow is injected through the syrinx not by the lungs, but by air sacs – another difference from mammals. The flow of air then sets the membranes in oscillatory motion to generate the sound [4]. Physically, this involves the subtle phenomena of negative damping. Both the glottis in humans and the syringeal membranes in songbirds are set in a self-sustained periodic motion not by the muscles themselves, but by an interplay between the restoring forces in the tissue and the dynamics of the pressure changes cause by the airflow [5]. As pointed out by [6], this driving force can be provided by Bernoulli forces and also by the presence of a traveling wave along the glottis. Although the two possibilities do not self-exclude, [2] shows that a number of bird species, including two songbirds, sustain the self-oscillatory motion of LM and MM via traveling waves in the tissue.

Direct observations of LL and MM in motion show how the degree of constriction can directly affect sound level [4]. It turns out that quieter vocalizations are produced by partially opening the syrinx and letting a portion of the air flow unmodulated. Louder vocalizations require a more constricted syrinx along with larger subsyringeal pressures. This also explains the mechanism for generating the fast amplitude modulations observed in some species. In fact, complete amplitude modulation can be achieved with only a slight opening of the syrinx [4].

This long list of observations helps us understand the variety of acoustic features present in birdsongs. But in order to claim that we fully understand the exact mechanisms, we must also be able to generate realistic birdsong syllables from simulations. For that, [7] constructs a model of the oscillatory motion of the syringeal membranes in analogy to earlier models of human phonation [6]. After coupling it to a filter representing the bird’s vocal tract, they show that it mimics canary syllables. This is done by controlling the tension on the membranes and the pressure at the bronchi. In fact, the small dimension
of the parameter space allows them to easily generate entire sequences of syllables by taking a continuous trajectory over the parameter space. But the space of vocal features which the syrinx can generate is much richer than just simple harmonic sounds. Some vocal features may arise directly from the anatomy of the organ. For example, fast transitions between periodic and aperiodic sounds, period doubling and mode locking have been replicated in vitro and linked to non-linear motion of the syringeal membranes [8]. Note, however, that these are effects observed by artificially manipulating the syrinx.

The coupling between the two sets of syringeal membranes and the bird’s vocal tract is also important for understanding the origin of song features. The typical assumption when studying vocalizations is to make a distinction between the source (the syrinx in birds and the larynx in humans) and the filter (the vocal tract cavity). The source-filter model assumes that sounds are produced by the source and its spectral content is shaped by the filter. The addition of filter-filter and filter-source non-linear interactions, including feedback from the filter back to the source, also predicts vocal effects such as subharmonic frequencies [9].

Humans control their vocal articulators, i.e. the glottis, pharynx, velum, tongue and lips, to produce speech sounds. By moving the articulators with specific gestures, the length and shape of the vocal tract can be modified to match the resonance peaks of different vowels. Songbirds can’t produce consonantal sounds in the same way humans do, but they use other articulatory movements to coordinate their vocal tracts to act as a filter, much similarly to how humans produce vowels [3].

In Fig. 1.1 we show spectrograms of speech and birdsongs from different species. We highlight relevant aspects for our discussions in the next chapters. In speech, formants, the resonant frequencies of the vocal tract, are the main carriers of vowel identity. Formants show as continuous bands of high intensity in the spectrogram. Consonants The song of the Bengalese finch starts with a bout of introductory notes. The bird then sings short motifs consisting of a few syllables separated by small gaps of silence [10]. The song is composed by sequences of motifs typically separated by filler notes and obey complex syntactic rules [11]. Despite their acoustic resemblance to filler notes, introductory notes don’t seem to play a role in the syntax. The zebra finch songs, on the other hand, has a fixed syntax. The bird repeats the same stereotyped sequence of harmonically rich syllables. The canary sings syllables in trilled repetitions in phrases. The phrase and its duration can follow complex transition rules [12].
Figure 1.1. Birdsongs and speech. All spectrograms are in the same temporal scale. Frequency axes differ. First column: a string of digits spoken by a female speaker (top), a portion of the song of a Bengalese finch (middle), and the song of a zebra finch (bottom). Second column: the beginning of a canary song (top) and the beginning of a song sparrow song.

1.1.2 Automatic recognition of birdsong syllables

Research in automatic birdsong recognition can be organized by two distinct problems: birdsong recognition and bird species recognition. In birdsong recognition, the goal is to identify vocal units produced by an unique individual during singing behavior. These units may be syllables, but in some cases entire song motifs are used as well. It is similar to speaker-dependent ASR, but with the distinction that the lexicon is unknown a priori. In research programs such as syntax modeling [11] and statistical learning [13] in songbirds, where interest concentrates on the sequences of vocal units, it may be enough for the system to predict solely the string of labels associated with the units sung. Speech-to-text systems typically accomplish this task. Many research programs, however, need to relate the acoustics of individual syllables or motifs with some behavioral, muscular or neural state of the bird at the moment of singing. These applications thus require recognition systems to predict not only the labels associated with each syllable or motif in the recording, but also their onset and offset times. In species recognition, on the other hand, the task is to identify unknown bird species from their recordings. It is particularly useful in bioacoustics and ecology. Although the two tasks share common techniques, their demands in terms of dataset and recognition accuracy are very different. Here we focus on birdsong syllable recognition.

Previous works have applied supervised techniques of ASR to birdsong syllable

In parallel, researchers have also developed tools for measuring song similarity. Quantifying the similarity of learned behavior is important for understanding how animals learn. [25] popularized computational methods for assessing song similarity and introduced many of the now standard tools in birdsong analysis. The task of measuring song similarity is closely related to isolated recognition in ASR, the difference being the final output. In recognition, the goal is to predict a semantic label, whereas in song similarity the goal is to rank a set of templates by some similarity score. [26] extended the method of [25] for continuous motifs. [27] used the mutual entropy to compare the distributions of average cross spectral densities of Bengalese finch and zebra finch syllables.

In developing measures for song similarity, researchers also expressed concerns about the possibility of introducing biases in the analysis. For example, zebra finches and canaries can discriminate between very fine structures of temporal resolutions below 1 ms in sound complexes, while humans cannot [28]. This raises whether some perceptual qualities commonly used to characterize sounds may not be available for songbirds and vice-versa. More recent works have used modern unsupervised techniques in hopes to avoid these potential biases and uncover latent representations of songs that describe them more compactly [29, 30]. These works also point out the importance of clustering as a pre-processing tool to assist with manual annotations.
1.2 Machine hearing

Machine hearing is sometimes taken as synonym to automatic speech recognition (ASR). ASR, however, deals with the problem of speech-to-text transcription. Machine hearing concerns applying ideas associated with hearing to machines. Although our goal in connecting birdsongs to speech is to simplify the problem of automatic recognition, we view the problem discussed in this dissertation as an instance of machine hearing.

ASR itself has been a field of intense research for decades. We can organize it by the subproblems it solves. For instance, in isolated recognition the boundaries of utterances are known a priori. This is in contrast to continuous speech recognition. In robust ASR, the goal is to develop systems immune to background noise degradation. Systems are also organized by their vocabulary sizes (small, medium, or large), speaker-dependent versus independent, conversational speech or read sentences, and others. In Ch. 2 we describe our initial attempts at robust ASR using the AURORA-2 dataset, a small vocabulary task [31].

Notable approaches in ASR, and in machine hearing, include dynamic time warping (DTW), hidden Markov models (HMMs), and deep neural networks (DNNs) and their variants. Over the last decade, DNN-based systems have consistently decreased the gap between human and machine performance in text-to-speech, including in adverse conditions [32]. DNNs thrive for large vocabulary tasks. Our goal here is not to attempt to outperform DNNs, but to provide an alternative solution. In particular, we aim to develop solutions that have the potential to be implemented in neuromorphic devices using spiking neural networks (SNNs). For the remaining of this section, we review some ideas from machine hearing and ASR which we will use throughout the rest of this dissertation.

1.2.1 Feature processing

Except for modern end-to-end models, learning algorithms in machine hearing operate on handcrafted features as inputs. These features are typically two-dimensional representations of the waveform in order to make periodicity explicit. The simplest examples are spectrograms, the magnitude of the short-time Fourier transform (STFT). Filterbanks are often applied to each time bin of the spectrogram to combine adjacent frequencies. Filterbanks also allow for rescaling the frequency axis and adjusting the bandwidths as a function of the frequency.

Other commonly used features are Mel-frequency cepstrum coefficients (MFCCs) [33]
and perceptual linear prediction coefficients (PLPs) [34]. In MFCCs, a filterbank in the mel scale (Fig. 1.2C) is first applied to the spectrogram. Next, the power spectrum in each time bin is passed through a logarithm, which effectively separates the contributions of vocal excitation and vocal tract filtering. Finally, the inverse FT is applied and the first few coefficients, which carry the formant information, are retained. PLPs are similar, but designed to match more closely the human hearing perception. It uses the Bark scale (Fig. 1.2C) along with a model of the critical bands of hearing in the filterbanks, an equal loudness curve to match the sensitivity of hearing at each frequency, and a power scaling of $1/3$. Additionally, it uses linear predictive coefficients (LPC) [35] instead of cepstral coefficients. Note that, in either case, there is no frequency axis.

The success of these handcrafted features in ASR systems suggest some important principles in their design. The use of sublinear scaling in the frequency axis is intended to match more closely perceptual judgments of pitch (Mel scale) and loudness (Bark scale). Strong compression is important for normalizing the dynamic range of the waveform. In speech, pitch carries the speaker identity, so separating it from the formants is important for speaker-independent recognition.

1.2.1.1 Gammatone filters

Gammatone filters have been popularized as linear approximations for the motion of the basilar membrane in the cochlea in response to sounds [36]. They also approximate the firing patterns of auditory nerve fibers after transduction by hair cells [37, 36]. Gammatone filters have been traditionally used in machine hearing due to their simplicity, even though they neglect the non-linearities in the cochlea. In particular, since the gammatone is a linear filter, its responses can be calculated by a simple convolution. Thus, it doesn’t suffer from the artifacts due finite-analysis windows which spectrograms typically use.

But perhaps the most striking argument in favor of gammatones comes from research in efficient auditory coding. [38] shows that linear filters, when optimized to collectively reconstruct a set of natural sounds, recover the exact shape of the auditory filters of the cat, even in the absence of any physiological information. Animal vocalizations or environmental sounds alone result in filters with mistuned bandwidths, suggesting that it is the statistics of natural sounds combined with the efficient coding hypothesis that shape the auditory filters. Indeed, gammatones are localized in time and frequency. This is important because natural sounds can range from a sudden thunder to the whistle of a songbird, and the auditory systems needs to represent efficiently both acoustic edges and smoothly varying sounds [39].
The impulse response of the $n$th order gammatone filter with center frequency $f$ is defined for $t > 0$ by
\[ g^n_f(t) = A(f)t^{n-1} \exp\{-2\pi b_ft\} \cos(2\pi ft). \] (1.1)

Here, $A(f)$ is a normalization factor and $b_f$ is a function that specifies the filter’s bandwidth. The gammatone impulse response oscillates inside an enveloping gamma function.

The equivalent rectangular bandwidth (ERB) approximation idealizes the auditory filters as rectangular bandpass filters. The model of [40] describes an approximation for this idealization at moderate frequencies using measurements from normal hearing young individuals:
\[ ERB(f_c) = 24.7 \times (4.37f_c + 1). \] (1.2)

In the equation above, $f_c$ is the center frequency of the auditory filter in KHz. The ERB scale of [41] allows us to make a parallel with other perceptual scales (Fig. 1.2C). The ERB scale is derived assuming that center frequencies are distributed along the hearing range in proportion to $ERB(f_c)$, so that the spacing between the filters is constant. [42] shows that for the gammatone filter of order $n = 4$, the ERB approximation gives the exponent $b_f = 1.024 \times ERB(f)$ in Eq. 1.1. We will use the $n = 4$ gammatone filter throughout this dissertation. We refer to the spectrogram produced by a $n = 4$ gammatone filterbank with the ERB approximation a cochleagram.

The $n = 4$ gammatone filter has a transfer function of order 8. [43] suggests two approximate implementations to reduce the order of the filter: a cascade of 4 filters of order 2 and an all-pole approximation of order 2. Both approximations reduce the running time, but introduce some artifacts in the output. We use an exact implementation of the filter instead, as we found it to be only slightly less efficient in practice (Fig. 1.2E).

In Fig. 1.2A we compare two cochleagrams of the same Bengalese finch syllable. For the left, we used a gammatone filterbank with center frequencies distributed linearly between 1 and 10 KHz and a constant ERB of 200 Hz for all the filters. For the right, we used center frequencies in an ERB scale derived from a linear fit to measurements of the auditory filters in European starlings [44]. We refer to it as the starling ERB. The ERB broadens and enhances the relative power in the bands around the fundamental frequency for this syllable. This, we speculate, reduces pitch variations across renditions of the same syllable, making it easier for classifiers to learn simpler decision boundaries. Fig. 1.2B compares the starling ERB to the ERB approximation of [40]. In songbirds,
the best hearing range falls between $2 - 5$ KHz and is limited to about 10 KHz [45]. Despite significant differences below 1 KHz, the two ERBs in Fig. 1.2B are very similar between $1 - 10$ KHz. Indeed, at close inspection, the cochleagrams generated by both filterbanks show negligible differences (not shown). Fig. 1.2D shows a filterbank of 16 gammatones tiling the range between $0.1 - 10$ KHz. The gain of the filters is normalized to 0 dB at the center frequency. For the rest of this dissertation, we will use gammatone filters in the starling ERB scale to process features. For all our results for Bengalese finches we use 32 filters and for the canary we use 48 filters.

### 1.2.1.2 Midlevel processing

We illustrate midlevel processing in Fig. 1.3. Fig. 1.3A shows the cochleagram of a Bengalese finch syllable generated by 32 filters (in constant bandwidths and in linear scale to facilitate visualization). The second row shows the band highlighted between the red lines. Note the large amount of detail in the fine structure of the cochleagram. The third row shows a half-wave rectified version of the same band. Half-wave rectification simulates phase-locking of auditory fibers. Finally, the last row shows the same band after half-wave
rectification and a strong compression of $1/100$. As we mentioned before, compression squeezes medium to high amplitudes around the same value, reducing dynamic range deviations and facilitating classification. In Fig. 1.3B we show the envelopes of the same band within 3 snippets of a Bengalese finch song. Note how the envelopes have very different amplitudes. In Fig. 1.3C we compare the envelopes of the same 3 snippets before and after compression. The drawback of strong compression is that it also enhances the amplitude of small perturbations, including noise.

Fine temporal scales are important in speech perception and can be perceived by songbirds. For example, the voice onset time is an important feature for the perception of voiced versus unvoiced stop consonants, such as /b/ and /p/ respectively. However, successful recognition of continuous speech also requires the listener to segment it. Segmentation requires a much coarser scale in order to ignore irrelevant details in the speech signal. Therefore, we retain only the envelope of the compressed bands. Fig. 1.3D
shows the temporal profile of the envelope of a single band throughout a Bengalese finch song. For our benchmarks, we fix the sampling rate of the envelope at 300 Hz.

Following envelope extraction, we normalize each band in mean and variance. We found in our experiments that whitening, that is mean-variance normalization, was detrimental to the performance when applied to test recordings. Instead, we decided to adapt each band by standardizing them in respect to the training set. That is, if $\mu_k$ and $\sigma_k$ are the mean and standard deviation of band $k$ from the training recordings, we transform the test spectrograms at band $k$ by $(S(t, f_k) - \mu_k)/\sigma_k$.

Finally, we also normalize each feature vector to lie on the unit sphere. Note that, since the system we describe in the next chapters integrate spectrograms over a short temporal duration, a feature vector corresponds to a small patch of spectrogram. Normalization then puts these patches on the surface of the unit sphere.

1.3 Overview

This dissertation is organized as follows. In Ch. 2 we introduce the concept of feature detectors and review our early attempts at continuous robust automatic speech recognition using spike patterns. We find the problem difficult to solve, even for a single speaker, due to the complexity of speech. To simplify it, we transition to birdsongs as a proxy for speech. In Ch. 3 we discuss continuous recognition of birdsong syllables of fixed duration using ultra-sparse spike sequences. We first discuss feature detector and template training. We then introduce a graph-based method for spike sequence reconstruction. We show that this method recognizes Bengalese finch syllables in continuous recordings using only a few seconds worth of syllables for training. In Ch. 4 we introduce modifications to the system for handling canary syllables. Differently from the Bengalese finch, canary syllables can warp in time and frequency. We show how these modifications improve the performance for Bengalese finch syllables. However, we find that most errors in canary songs stem from misclassifications. To fix this problem, we introduce the concept of binary detectors and show how it reduces all error rates. Finally, we compare our results with the literature. In Ch. 5, motivated by the song of the song sparrow, we discuss the problem of compositionality, where syllables of different types can share subunits such as notes. We discuss how the note repertoire can be constructed using feature detectors to help in recognition. We then introduce the idea of spike clustering to help with song and syllable segmentation. In Ch. 6 we switch topics and describe a method for semi-automatic reconstruction of neuronal somata using slice images. We test it using
images of neurons from the mouse’s V1. Finally, we show how our reconstructed surface areas compare to the reported values.
Chapter 2
Speech recognition with spike sequences

In this chapter, we describe our early attempts at extending the isolated spoken digit recognition system of [46] to the continuous domain. Our main goal is to introduce some important concepts and motivate our work for the following chapters on birdsong syllable recognition. We start with a discussion on the representation of speech as sequences of spikes. We then introduce the concept of feature detectors. Next, we review the system of [46] and elaborate a plan for extending it using rapid spectral changes in speech as temporal landmarks for segmentation. We show that insertions become our main difficulty when background noise is added. We conclude the chapter addressing the difficulties of continuous speech recognition using spike representations. To circumvent these difficulties, we propose a simpler but analogous problem: birdsong syllable recognition.

2.1 Spike-based representation of speech

Why represent speech at the level of spikes? In the natural environment, communication sounds are necessarily embedded in an acoustic scene. This scene can include sounds ranging from the constant burbling of a river to the chatter of a group of people. Auditory systems have evolved to attend, extract, and make sense of target sounds from acoustic scenes. This is known as the auditory scene analysis (ASA) problem [47]. The matter of fact is that animals solve this complicated computational task with ease. More recent research has delved into how spike codes may solve the ASA problem in the brain, but the precise form of computation used by auditory systems is unknown. One hypothesis is that temporally and spatially sparse and precise auditory codes represent target sounds.
in the brain [48, 49, 50, 51]. In the avian brain, for instance, higher auditory codes have been found to robustly represent conspecific vocalizations embedded in complex acoustic scenes [1]. Together, these findings suggest that biological systems may use robust, efficient, and sparse auditory codes to solve ASA.

2.1.1 The STRF

We start with a brief discussion of the concept of the spectrotemporal receptive field (STRF), which helps us connect our ideas to the literature in auditory neuroscience. The STRF is the linear transfer function which relates some spectrotemporal input \( s(t, f) \) to some measured output trace, \( r(t) \), typically the firing rate or the subthreshold voltage of a neural system:

\[
    r(t) = \int df \int d\tau h(\tau, f)s(t - \tau, f) + r_0. \tag{2.1}
\]

Here \( r_0 \) is a constant firing or bias term. STRFs are useful for characterizing auditory neurons. By visually inspecting the STRF we can infer the linear contributions of the neuron’s response at specific frequency and lag values. Real neurons are not linear and time-independent systems, but the STRF helps with understanding the linear portion of firing patterns. In practice, we discretize Eq. 2.1 in time and frequency. We assume that \( h \) has a short lag (that is, the response of the filter’s impulse is finite) and that it drops to zero everywhere else. As we will see in more detail in the following chapter, we can vectorize matrix \( h \) by concatenating all rows together. That is, \( h \) is a stack of vectors, each spanning the frequency axis and with a fixed lag value. If we do the same to the input \( s(t) \), we can write \( r(t) = h \cdot s(t) + r_0 \). Finally, a common way to introduce non-linearities and retain the STRF is to apply some non-linear activation function \( f \) to the response \( r \),

\[
    y(t) = f(r(t)) = f(h \cdot s(t) + r_0). \tag{2.2}
\]

The simplest possible non-linearity \( f \) we can use is a threshold function that returns 0 everywhere except where \( r(t) > 0 \). One way to introduce spikes in our discussion is by assigning them at the points where \( r(t) > 0 \). In this case, \( r_0 \) represents the threshold that the voltage trace \( h \cdot s \) has to surpass. A more complicated, but computationally simple and more realistic, activation function \( f \) would also introduce adaptation and a refractory period. Note that training \( h \) to some stimuli amounts to the same as defining the STRF of the system. Geometrically, \( h \) defines a hyperplane that separates inputs with \( r(t) > 0 \) from those with \( r(t) < 0 \).

In our discussions, we will interpret \( r(t) \) as the subthreshold voltage of a neuron. We
will also depart from the simple threshold function $f$ and assign spike times not at the crossing of the threshold but at the peak of $h \cdot s$ after the threshold is crossed. The question we now ask is how can we tune $h$ for recognizing features in spoken words. Can we tune it in such a way that Eq. 2.2 produce precise and robust spike patterns capable to representing the correct stimuli even in the presence of noise?

2.1.2 Feature detectors

In our use of the term, feature detectors are units that identify specific patterns in the input. That is, they classify points in the input space into two types (1 or $-1$, 1 or 0, positive or negative, etc). Here we are interested in spectrotemporal patterns within vocalizations. These are acoustic events of finite duration and limited bandwidth, so we will use the term feature or feature vectors to refer specifically to patches of the spectrogram. In order to identify these patches, feature detectors need to learn suitable values for $h$ in Eq. 2.2.

One simple way to tune $h$ for feature detector learning is using the perceptron. The perceptron algorithm updates its weights whenever the input pattern is misclassified. Geometrically, this corresponds to rotating and translating the hyperplane represented by $h$ so that it separates the two classes of patterns. Note that binary classification can also refer to separating a class of patterns versus the rest of the set of stimuli, which is the concept that we use here for detecting features. If these classes of patterns are linearly separable, the perceptron converges in finite many steps. What is interesting to pointing out is that if the vectors are all bounded within a sphere of radius $R$ and the euclidean distance from the hyperplane $h$ to the closest vector is $m$, the convergence happens in at most $(R/m)^2$ steps. This number, $m$, is known as the margin.

In support vector machines (SVMs), the hyperplane $h$ is optimized by margin maximization [CITE]. Thus, SVMs fall under the category of maximum margin classifiers. This leads to a constrained optimization problem which has a unique solution when the training patterns are linearly separable. Namely, the hard margin SVM minimizes $|d|^2$ under the constraint that $y_i r(t_i) \geq 1$ for all patterns $t_i$ with label $y_i$ (conversely, it maximizes $1/|d|^2$, which is the size of the margin). Vectors satisfying $y_i r(t_i) = 1$ are called support vectors and lie on the margin. The SVM solution is a sparse linear combination of the support vectors. The optimized hyperplane can be written as

$$h = \sum_k \alpha_k h_k,$$  \hspace{1cm} (2.3)
where $\alpha_k$ are given by the SVM optimization and are only different from zero for the support vectors. Here we will be concerned with hard margin SVMs only, where separability is assumed. This assumption does not necessarily hold for short snippets or speech or birdsong syllables. For example, it is known that vowel in the formant space largely overlap. We will discuss how to circumvent this issue in the context of birdsong syllables in the next chapters. Note that, just like the perceptron, the SVM can be also optimized via gradient descent. Namely, the SVM can be introduced as minimizing the hinge function regularized by an L2 term:

$$L = \frac{1}{2} |h|^2 + C \sum_i \max(0, 1 - y_i r(t_i)).$$

We will use this approach when we discuss boosted binary detectors in Ch. 4. The constant $C$ balances the regularization term with the hinge loss. This is a departure from the hard margin approach and allows some patterns to be non-separable during training.

### 2.1.3 Sequences of spiking detectors

If we have a pool of trained feature detectors, we can easily encode utterances as sequences of event detections. If the utterance is embedded in some input $s(t)$, we can simply pass $s(t)$ through the pool of detectors (Eq. 2.2) and record their activity. When the activation $f$ is a hard threshold with a refractory period (or, as we will use in the next section, a routine that places spike times at the peaks of the subthreshold voltage trace $r(t)$), the input is effectively transformed into a spike raster. In Sec. 2.2.1 we order this raster by spike time and keep only the sequence of indices of the units activated. However, in the following sections, we keep the spike time.

### 2.2 Speech recognition using spike sequences

Representing speech sounds as sequences of spikes is a difficult problem because speech can be very flexible. What aspects of the acoustic signal are essential for comprehension? What features can be ignored? Are the lexical aspects essential? The representation of speech in the brain is also difficult to study at fine spatiotemporal scales because it requires invasive techniques. Recent studies using high-density electrocorticography have shown evidence that non-primary auditory cortical areas represents and restores attended speech from mixed speakers [52]. The superior temporal gyrus rapidly provides access to a spatial representation of phonetic features [53]. This representation presumably
allows the brain to categorize speech sounds such as phonemes and syllables, but the exact mechanism is unknown. Nevertheless, these studies motivate our spike-based representation of speech.

Inspired by these and other studies, [46] proposed a spike-based system for robust isolated digit recognition. In this section, we review this system and its limitations. We then introduce modifications in an attempt to extend it to the continuous domain.

2.2.1 Isolated recognition with spike sequences

The system introduced by [46] recognizes isolated spoken digits by template matching. Each template represents a spoken word in the vocabulary (in this case the digits ‘zero’ to ‘nine’ and also ‘oh’) using a string of feature detector ids. To capture speaker-related variations in speech, the system uses about 100 templates per digit, each trained on a different speaker. This includes 50 female and 50 male speakers, as gender is one of the main sources of variability in spoken words. The order of ids in each string reflects the firing pattern of the pool of feature detectors. Note that the templates do not carry spike times or relative spike times, only the order of firing. Thus, for decoding, only the relative order is compared to the input. This is important because the system uses the length of the longest common subsequence (LCS) to score the input’s string against the template. The LCS skips insertions in the input’s string, which, for this system, introduces robustness when background noises drive feature detectors to fire spuriously.

To effectively compare the LCS lengths between digits of different duration (for most speakers in the AURORA-2 dataset, for instance, ‘oh’ is typically much shorter than ‘seven’), their system uses a z-score computed by sampling random patterns of strings and matching them against the templates. Finally, the system uses an n-best decision rule where the winning category is chosen from the averages of the n-largest scores within each template category. In summary, the score assigned by matching some sequence $S$ against a template $T$ is given by $|\text{LCS}(S,T) - \mu_T|/\sigma_T$, where $\mu_T$ and $\sigma_T$ are the mean and standard deviation of matching $T$ against random strings. The LCS score seems to combine well with the SVM’s lower margin. Recall that the purpose of the margin is to leave some padding around the decision boundary in order to help with generalization. By using the SVM’s lower margin for detection, the system is more prone to, at the same time, introduce insertions from false positives and reduce deletions from true positives. The negative effect of insertions is counterbalanced by the LCS score, while the positive effect from deletion reduction remains.

We benchmarked again this system using the AURORA-2 dataset [31]. Fig. 2.1A
Figure 2.1. Recognition of isolated spoken digits under noisy conditions using spike sequences. (A) Score matrix for the Test A set in the AURORA-2 task under clean conditions. Each row represents a test utterance and each column represents a template. Color indicates the score. The block diagonal shows correct recognition. (B) Accuracy for the same task as in panel A using an N-best average scheme to make predictions. (C) Two sequences of the digit ‘eight’ produced by different speakers and under different SNRs of subway background noise. (D) Performance of the spike-based recognition system to all tasks in the AURORA-2 dataset for isolated digits. The system’s performance falls short from listeners recognizing connected digits [54].

illustrates the typical score matrix for this task. Its block structure makes it explicit that templates tend to score higher to their preferred digits. At the same time, the bright non-diagonal blocks reflect that templates are speaker dependent and may sometimes also score high to negative digits. This justifies why averaging over many templates helps with recognition. For instance, Fig. 2.1B shows the performance of the system across multiple n values for the n-best averaging. In Fig. 2.1C we illustrate two spike sequences representing the word ‘eight’ and sorted by the relative spike order in a single template. We show how the sequences degrades in the presence of subway background noise at different intensities. Note how the overall patterns are still visible at 10 dB, while insertions and deletions blur them out.

In Fig. 2.1D we show benchmarks for the entire AURORA-2 task, including different
noise conditions and levels. The last panel of Fig. 2.1D simulates the filtering of a telephone line [31]. We also show how it compares to the performance of adult listeners [54] for the same data with the exception that subjects listened to continuous strings of digits. More recent evaluations of the recognition gap between listeners and ASR systems have suggested that it has decreased dramatically with the introduction of modern DNN-based systems [32].

In going forward with continuous recognition using spike sequences, what are the limitations of the LCS method? Can we patch these limitations and achieve the same level of robustness for connected digits?

### 2.2.2 Towards continuous recognition with spike sequences

Why is the LCS method hard to use for continuous recognition? Continuous recognition by template matching requires stitching together sequences of templates in order to match the input. Differently from DTW, LCS supports arbitrary amounts of stretching between the two sequences. That is, there is no penalty when symbols are skipped in the LCS. Thus, a direct application of LCS for continuous recognition will jump through multiple templates while selecting the best set of matching frames. It may, for example, select a template for each frame. An alternative way to solve continuous recognition is by first segmenting the input at specific temporal landmarks. This idea has support from modern theories of speech processing in the brain [55].

#### 2.2.2.1 Segmentation using rapid spectral changes

How can we segment speech? Previous research in our group showed that it can be done using landmark features in the syllabic onset and nuclei. However, detection of these features is not always accurate, and the performance degrades rapidly in the presence of background noise. Errors in segmentation can have a large impact in the final recognition accuracy of the system. So one question we tried to address was whether we could train feature detectors to spot landmark features in speech more robustly.

First we asked if we could train feature detectors to respond reliably to vowels in speech. The identity of isolated vowels depend on their formant values, which can be ambiguous for discrimination [56]. Indeed, when we trained feature detectors to discriminate between snippets of vowels, we found them to be unreliable. In casual speech, transitions between formants are common and rarely reach their target frequency values. Perception of vowel categories cannot be solely based on formant target resonance.
Figure 2.2. Feature detectors for CV transitions. (A) Short spectrotemporal patches around F to AO transitions from a single male speaker. (B) Feature detectors can robustly spot in synchrony CV transitions in continuous speech. Below the spike raster we show the PSTHs. (C) The same feature detectors as in panel B can generalize to other male speakers. (D) STAs of two CV transition detectors. Left: F to AO transition. Right: S to EH transition.

frequencies [57]. Alternatively, consonant-vowel (CV) transitions seem to contribute to the perception of vowel identity [58]. The portions containing the largest amount of spectral change seem to be particularly important for perception [59]. This prompted us to train feature detectors for portions of large spectral transitions in the digits.

We first focused on a single speaker and used hand-segmented phonetic transcriptions to select the segments. Although we observed that sometimes detectors would respond more often to one particular CV than others, we found them largely unreliable due to their false positive rates. This makes sense because we shouldn’t expect speech sounds to be linearly separable in the feature space. In [46], feature detectors are not trained to respond to particular phonetic feature, but to discriminate against snippets of speech randomly pooled from spoken digits. Can we effectively train feature detecting units for phonetic features with the help of non-linearity?

In Fig. 2.2A we show segmented snippets of 80 ms centered around the transition between F and AO in the digit ‘four’ from a single speaker. Here we use the phonetic notation of the CMU Pronouncing Dictionary. Note the similarity between the spectro-
grams and the sharp transition between the F and the AO sounds. We then trained non-linear SVMs to discriminate between a subset of these snippets against all other phonetic features in digits from same speaker. One important distinction here is that we used the decision boundary for detection. In Fig. 2.2B we show the responses of detectors to a string of connected digits from the same speaker. We found that these detectors generalized for other male speakers as well (Fig. 2.2), although they tended to be not very reliable for female speakers. In Fig. 2.2D we show the spike triggered averages (STAs) of two feature detectors trained in this manner. STAs estimate the STRF by averaging the all the stimuli which prompts the unit to respond. The first STA correspond to the detectors shown in panels B and C. The second STA corresponds to a CV transition from S to EH, as in the digit ‘seven’.

Can this be generalized to other CV transitions? F is a fricative consonant. Would it also work, for example, for a plosive? We tried the transition from T to UW, as in ‘two’ (Fig. 2.3A). In Fig. 2.3B we show the responses of all 8 units trained when the decision boundary (top) and the lower margin (bottom) are used for detection. The two figures suggest a trade-off between deletions and insertions. In Fig. 2.3C we show the counts of detection during testing. Note the suspiciously high confusion between digits ‘two’ and ‘six’. Inspecting these detectors, we found that many negative support vectors were actually S to IH, as in ‘six’, transitions (Fig. 2.3D). It indicates that the two classes are adjacent in the feature space. By removing these negative support vectors during training, the confusion between T-UW and S-IH increases (Fig. 2.3E). In Fig. 2.3F we show the responses of a T-UW detector before (in black) and after (in red) support vectors from other similar transitions are removed from the training set. The dashed lines display the lower margin. Note how the red margin is larger and how the red curve peaks higher. Note also how it responds vigorously to both ‘two’ and ‘six’. In fact, this is a detector that covers both classes.

We then repeated this process for all digits. Here we also included transitions from silence, denoted by SIL, to vowels. For example, SIL to EY, as in ‘eight’. We emphasize that our intention is not to detect specific classes of phonetic transitions. Instead, we aim to carve the feature space into separated regions that can be robustly identified in continuous speech. For example, even though the first formants in UW and IH are similar, the second and third formants are very different. So, acoustically, it’s not true that IH and UW are similar. Fig 2.4A shows a short snippet of a subset of our detectors. There are many deletions, but the detectors, collectively, cover the entire sequence of digits. However, to recognize the utterances, we need to discriminate between, for example, ‘two’
Figure 2.3. Feature detectors for groups of CV transitions. (A) Short snippets surrounding the CV transition T-UW, as in ‘two’, from a single male speaker. (B) Using the snippets in A, we train detectors to recognize the transition T-UW. The vertical red lines correspond to detections during testing. The ground truth is shown in blue. The detection threshold trades deletions when placed at the decision boundary (top) for insertions when placed at the lower margin (bottom). (C) Detection counts during recognition. Most confusions come from the digit ‘six’. (D) Example of 12 negative support vectors from one of the detectors. Many of them represent the CV transition S-IH, as in ‘six’. (E) When S-IH is removed from the negative training set, confusions with the digit ‘six’ increase. (F) Response of a detector tuned for T-UW which doesn’t discriminate against S-IH. While the confusions between ‘two’ and ‘six’ increase, the detectors’ responses become more robust.

and ‘six’.

For this, we trained another set of feature detectors to discriminate between the CV transitions in ‘two’ and ‘six’. A major distinction here is that these units are specialized. They only need to discriminate between two transitions. This idea is a precursor of a concept that we will formally introduce in Ch. 4 and which we call binary detector. We also carried the same idea for the other classes of CV transitions, including silence to vowel transitions (see the y-axis in Fig. 2.4A). In Fig. 2.4B we show the system for a single speaker during testing. The first panel shows the confusion matrix between the transition classes. The second panel shows detection counts for each pool of detectors in respect to the digit classes. In both cases, the threshold is set at the decision boundary.

Now we ask if this system retains its accuracy in the presence of noise. Here we
also need to account for insertions during silent pauses (SP). In Fig. 2.4C we show the performance broken down into true detection, insertions, and SP insertions in all noise conditions. While the true detection rate degrades slowly, the rate of insertion grows very rapidly.

### 2.2.2.2 Recognition using relative delays

Once the input is segmented, we need to properly classify the patterns contained within each segment. In principle, we could use the LCS method now that we’ve found precise temporal landmarks for segmentation. However, in the absence of other landmarks to indicate the end of each digit, we would need to create templates representing the segments between all possible tuples of onset transitions. For example, to recognize a


Figure 2.5. Using relative spike delays for isolated recognition. (A) Number of times each template is found in the n-best (here 5-best) list throughout the benchmark as a function of the number of spikes they contain. Larger templates are more likely to be chosen. (B) Benchmark on isolated digit recognition using PSTH method (synchronous), the PSTH method when templates are normalized (synchronous, norm), and the LCS method.

‘two’ followed by a ‘six’, we would need templates representing the acoustics between the transition T-UW and the transition S-IH. Another drawback of the LCS is that it can be slow in practice. Finally, it is not clear how it could implement in a network of spiking neurons.

To circumvent these difficulties, we asked if we could use the relative delays between spikes in the templates. This can be easily implemented in a spiking network with the introduction of axonal delays. If the spike sequence synchronizes after it goes through the axonal delays, it means that the delays matched the order of firing in the sequence. We will explore this idea in more depth in the next chapters.

We implemented it using linear SVMs and the lower margin for detection. In practice, we convolve the spike trains with gaussian bumps of small widths to simulate the firing of a larger pool of units. In Fig. 2.5A we show that templates with larger numbers of spikes tend to appear more frequently in the n-best list during decoding. To account for template length, we used the relative spike delays to score sequences sampled at random and calculated their z-scores, as we explained in section 2.2.1. Fig. 2.5B shows that even when the templates are normalized, this synchronization method still falls short from the LCS in the isolated case.

From our results in Figs. 2.5B and 2.4C, it is clear that the two methods combined will perform below the LCS’s accuracy in isolated recognition. Figs. 2.4C-D also suggest that in continuous recognition our main difficulty should be in dealing with insertions. Our discussion only covers a single speaker. To extend our methods for speaker-invariant recognition, we must repeat the training procedure for multiple speakers. However, this
is difficult to implement without an automated training procedure.

2.3 Conclusions

In this chapter, we introduced the concept of feature detector of spectrotemporal patterns in speech. We briefly reviewed a spike-based system for robust recognition of isolated spoken digits and explored ways to extend it to the continuous domain. We found detectors trained to recognize speech sounds containing fast spectral changes to be the most reliable. However, we found the problem of balancing false positives and false negatives during recognition difficult even for a single speaker.

In going forward, we decided to simplify the problem. In the following chapters, we will focus on birdsong syllable recognition using spike sequences. Birdsongs, although acoustically much simpler, resemble speech in a few basic aspects and will serve us as test beds for a spike-based continuous recognition system.
Chapter 3
Continuous recognition of vocalizations of fixed-duration

In this chapter, we develop a framework for continuous recognition of birdsong syllables using ultra-sparse sequences of event detections. To simplify the problem, we assume that vocalization classes have fixed durations. This assumption does not hold in general for speech or for the vocalizations of most animals. However, in some songbird species, song stereotypy is a marked trait. In zebra finches, stereotypy is important in courtship behavior [60]. Bengalese finch syllables also show remarkable duration stereotypy. Note, however, that tempo in birdsongs can show circadian and age-related changes [61]. We do not discuss these effects here.

We start in Sec. 3.1 with a brief discussion of syllables of fixed duration. In Sec. 3.2 we revise the concept of feature detector introduced in the previous chapter and discuss detector and template training. In Sec. 3.3, we discuss continuous decoding of spike sequences. We show how simple delay lines support rapid detection and formation of an auditory scene. When insertions and deletions degrade the spike code, we’re forced to adopt a more detailed method for recognition. For this, we try estimating the distributions of relative delays between spikes within sequences, but find it difficult to automate in the regime of small training sets. We then switch, in section 3.3.5, to a graph-based approach for sequence reconstruction. To assist with our exposition, we introduce concepts such as the spike-alignment graph (Sec. 3.3.5.1), sequence completion (Sec. 3.3.5.2), and the prediction stream (Sec. 3.3.5.3). In Sec. 3.3.6 we experiment with direct manipulation of feature vectors. Finally, in Sec. 3.3.7, we show that our ideas support recognition of Bengalese finch syllables using as little as 5 exemplars per syllable type for training. We conclude addressing the limitations of our system and pave the way for extending it in the following chapters.
Figure 3.1. Temporal warping in a Bengalese finch syllable. (A) The four renditions of this Bengalese finch syllable have the exact same duration. However, small variations in the acoustics are evident. (B) Distribution of durations of 100 syllables of the same type as in panel A. Duration variability is negligible. (C) Example of DTW alignment between two syllable exemplars. (D) DTW distortion for this syllable type concentrates around zero. Distortion is defined as the distance between the alignment and the identity line.

3.1 Syllables of fixed-duration

The assumption that every rendition of a vocalization type has a fixed duration simplifies the recognition problem. If the duration is fixed (or narrowly distributed), seeing one exemplar is sufficient for inferring the duration of any new ones. For continuous recognition, this is extremely helpful because it allows us to predict the end of the syllable right after we detected its onset. But it doesn’t completely eliminate all difficulties coming from variability in the acoustic signal. Even if two vocal elements have similar power spectra and identical durations, they may still have completely unrelated temporal profiles. Such cases are readily seen in Bengalese finch syllables. For example, compare the 4 syllables in Fig. 3.1A. But even in speech, changes in rhythm can stretch a vowel around the middle portion while compressing it on the edges, thus maintaining the overall duration. Formants in speech can vary in shape and frequency without altering the vowel’s category. So, clearly, a single quantity, duration or frequency content, is not enough to characterize it. That said, due to the incredibly narrow duration distributions of Bengalese finch syllables, some classes of syllable can sometimes be discriminated based on duration only.

It’s illuminating to compare temporal warping in syllables of different species. The canary syllable in Fig. 3.2A, for instance, warps by factors larger than 3 (Fig. 3.2B, 100 randomly selected exemplars). Note the similarities in the initial pitch and in the target pitch between the renditions. The initial upward sweep varies little between the 4 examples, but the following stationary harmonic stacks have very different durations. Since canaries are known to sing in phrases of repeated syllables, we know that these
Figure 3.2. Temporal warping in a canary syllable. (A) Four renditions of a canary syllable with variable duration. (B) Distribution of durations of 100 syllables of the same type over multiple songs. (C) DTW alignment between two syllables of the same type. Note the difference in duration and how most of the warping occurs in the mid-portion. This suggests that duration changes occur in the flat harmonic stack. (D) Distribution of DTW distortion for the same syllable type. Distortion is defined as the distance between the alignment and the identity line.

Figure 3.3. Comparing acoustic distortion and duration in the canary and in the Bengalese finch repertoire. (A) and (B) Width of the distributions of syllable duration and distortion. Bengalese finch syllables are comparatively much more stereotyped.

syllables belong to the same category. However, as the DTW solution in panel 3.2C suggests, both the initial sweep and the flat harmonic may show some amount of temporal distortion. DTW distortion in panel 3.2D is defined as the pointwise distance between the DTW solution and the identity line, and is normalized by the duration of the DTW template.

Compare the canary syllable to the four randomly selected renditions of the Bengalese finch syllable shown in Fig. 3.1A. Bengalese finch syllables are remarkably stereotyped in spectrum and in duration (Fig. 3.1B, 100 randomly selected exemplars). DTW alignments for Bengalese finch syllables don’t typically deviate much from the identity, but in panel 3.1C we show a rare example to contrast with the canary syllable. The narrow distribution of DTW distortions in panel 3.1D, as opposed to the canary syllable, suggests that variations between renditions, in the absence of differences in duration, come from small acoustic variations in pitch and spectral contour.

All Bengalese finch syllables (excluding calls and introductory notes) we study here
have narrow duration distributions and negligible temporal distortions. We summarize this fact by showing in Fig. 3.3A the widths of the distributions of DTW distortions from all syllable types in 8 Bengalese finches (in blue). To contrast, we also show a single canary (in red). Note that most of the syllables sung by this canary show only small amounts of temporal warping, except for one. Fig. 3.3B summarizes the width of the duration distributions for all birds. Although not shown here, zebra finch songs are also extremely stereotyped in duration [62]. However, the syntax of the zebra finch song is much simpler than of the Bengalese finch. This motivates us to use the Bengalese finch song in this section.

### 3.2 Spike-based representation of syllables

In Ch. 2 we discussed the concept of acoustic feature detectors and how we can use it to construct a spike-based representation of speech. In this section we develop this concept further for encoding syllables as ultra-sparse sequences of spikes. The problem we solve here is the transformation of the spectrogram into a representation formed by short acoustic events that is also selective for syllable types.

Our main assumption is that we can represent syllables as sequences of short-time acoustic segments. This idea is reminiscent of how state models like hidden Markov models (HMMs) represent speech in ASR [63]. In HMMs, states are trained to represent probability distributions of observed short-time segments in utterances. At the same time, transition probabilities are trained to force the model to jump from one state to another according to the sequence of observations. To represent a word, for example, an HMM may be built by connecting sequences of states representing the word’s phonetic transcription. The training algorithm alternates between generating a label sequence (an alignment) for each frame of audio and using these labels to make predictions. This process iteratively tunes the states’ parameters to the statistics. It automatically solves a difficult “chicken or egg” problem: without an initial reliable alignment, how do we decide when to jump from one state to another? And, without a reliable model of the data, how do we generate a precise alignment? To circumvent this problem, here we focus only on discriminative models.

One immediate question that the idea of representing vocal units as sequences of short segments raises is: how short is a short segment? While in practice the best range of durations should depend on the acoustic properties of the vocal repertoire, we speculate that it can be neither too long nor too short. If the segment is too short, there will be a
3.2.1 Syllables as ultra-sparse spike sequences

To simplify the problem, we select a fixed short-time duration $T$ for all syllable types. For the Bengalese finch, we use $T = 35$ ms. However, we speculate that a range of short-time durations may improve recognition when allowed to fine-tune to the statistics of the syllable classes.

So suppose we take a fixed-length window of size $F \times T$ and slide it along the temporal axis of the spectrogram of a syllable. We assume that $F$ spans the entire frequency axis and $T$ is shorter than the syllable. If we vectorize this $F \times T$ matrix, we end up in an $FT$-dimensional vector space whose points we call feature vectors (we also refer to this vector space as a feature space). As we unroll the window along the temporal axis, the feature vectors describe a curve in the vector space, as illustrated in Fig. 3.4A. If the syllable is short, it may be described by a single feature vector. This way of slicing the spectrogram is common in STRF learning, where dimensional reduction, usually PCA, is usually applied to speed up learning [66, 51]. SVMs benefit from higher dimensions, so we do not use dimensional reduction.

Next, we repeat this process with the other syllables in the same category. The curves generated will coincide with the first one. Feature vectors representing similar acoustic features will cluster around specific regions of the feature space. To encode an acoustic event represented by a feature vector $x^+$, we use the cloud of points surrounding it to construct a decision boundary. This decision boundary needs to isolate $x^+$ from all other distinct feature vectors. But, more than that, it also needs to enclose and group together acoustic features similar to $x^+$. We’ll call the vectors enclosed by the decision region positive and denote them by $x_i^+$. Similarly, we’ll call the rest of the feature vectors negative and denote them by $x_i^-$. Thus, we can look at this problem as a binary classification task. This is illustrated in Fig. 3.4B, where the decision boundary encloses a region containing the start of a syllable and excludes all the rest. Note how we avoided the “chicken-and-egg” problem by selecting an initial syllable as a template. This initial syllable provides the feature vector $x^+$, while the rest of the positive vectors
Figure 3.4. Encoding short-time acoustic events in birdsong syllables. (A) The spectrogram of a syllable is chopped into short-time segments which are then mapped onto a vector space. The progression of these points in the vector space traces a path representing the sequence of segments that compose the syllable. (B) Points representing the same acoustic event (positive feature vectors) cluster in the feature space. A decision boundary encodes the acoustic event by separating the positive points within it from all the rest (negative feature vectors).

will need to be extracted from other syllables in the same category. We refer to it as the seed syllable. As we will see in Ch. 4, this strategy may introduce issues if the seed is poorly selected.

In the linear case, the decision boundary is a hyperplane parametrized by some weight vector $w$ and bias $b$. If $|w| = 1$, for any $x$ the point to plane equation is $w \cdot x + b$. Given a parametrized curve $x(t)$ in the feature space, the distance to the decision boundary at each instant is $y(t) = w \cdot x(t) + b$. Now assume that the weight vector can be written as a sparse linear combination of the positive vectors, $w = \sum_i \alpha_i x_i^+$. Then,

$$y(t) = \sum_i \alpha_i x_i^+ \cdot x(t) + b \ . \quad (3.1)$$

Going one step further, we assume that the feature vectors are obtained by mapping the $F \times T$ windows onto a feature space defined by some transformation $\phi(x)$. The dot product in Eq. 3.1 becomes $\phi(x_i) \cdot \phi(x(t))$. We define the kernel $K(x,y) = \phi(x) \cdot \phi(y)$ and write Eq. 3.1 as

$$y(t) = \sum_i \alpha_i K(x_i^+, x(t)) + b \ . \quad (3.2)$$
For an arbitrary $K$, the decomposition into an inner product may not be possible. However, under the assumption that $K$ is non-negative and symmetric, we can write $K(x, y) = \phi(x) \cdot \phi(y)$ for some $\phi$. This is the case, for instance, for the Gaussian kernel:

$$
k(x, y) = \exp\{-|x - y|^2/2\sigma^2\}.
$$

(3.3)

Figure 3.5A shows $y(t)$ for two acoustic events encoded using a Gaussian kernel. The offset $b$ is shown by the dashed line. The trace on the top figure peaks for a downward harmonic sweep and remains below the offset for all other vocalizations. The bottom figure shows a sustained response to a flat and high-pitched harmonic stack. Note how it briefly peaks just above the offset for the following harmonic stack. This is expected as the beginning of the two stacks are almost identical and represent the same acoustic event.

Now suppose we carry this procedure for a subset of all $F \times T$ windows of all spectrograms. Each trained decision boundary may span only a small portion of the curve $x(t)$, particularly when the syllable is long. But, collectively, the set of decision boundaries may cover the entire path and provide a transformation from the spectrogram to a sequence of detections. Fig. 3.5 breaks this idea down for a syllable using 6 decision boundaries. The responses of the decision boundaries, Eq. 3.2, are illustrated along syllable’s path in the feature space. Note that they overlap in time. We group these responses $y_i$ in $N$-tuples of the form $Y(t) = (y_1, \ldots, y_N(t))$, $N$ being the number of decision boundaries. We assume that the ordering in $Y$ reflects the order in which acoustic features were selected in the seed syllable.

The representation $Y(t)$ may not be sparse, but note how each $y_i$ response crosses the offset and peaks at a distinct time. The time of each peak reflects the instant of maximal similarity between the song and the acoustic feature. We’ll refer interchangeably to the decision regions as feature detectors and to the offset $b$ as the detector’s threshold. To sparsify $Y$ we can binarize it by assigning 1’s to peak times above the threshold and 0’s everywhere else. To implement it in practice, first we clip the responses above the threshold:

$$
z_i(t) = \max(0, y_i(t) - b_i).
$$

(3.4)

For each contiguous non-zero support of $z_i(t)$, we find the corresponding peak times. At this point we can adopt some different strategies. We can assign 1’s to all peak times. Or we can assign 1’s to the highest peak only. We can, as before, also remove adjacent peaks from small fluctuations or impose a refractory period. In any case, the operation
applied at this step may depend on a set of time points and the corresponding values of $z_i$. We’ll summarize these operations in a map $F$ and write the binary representation as

$$s_i(t) = F(z_i(t - n\Delta T/2), \ldots, z_i(t + n\Delta T/2)),$$  

(3.5)

where $n = 0, 1, \ldots, N$, $\Delta T = 1/f_s$ is the frame duration ($f_s$ is the sampling rate) and $n\Delta T$ is the duration of a context window. The specific operation we implement in $F$ will dictate the $N$ we use. Fig. 3.6A illustrates the construction of $s_i(t)$ with a refraction duration of 100 ms. Note that the second peak in $z_i(t)$ is ignored. Fig. 3.6B shows 3 syllables encoded in the $Z$ representation. These are syllables used during training. We will call them, along with their $Z$ and $S$ responses, templates. Note how the acoustics is reflected in the amplitudes of the consecutive responses. The start of the syllable is a moment of transition from silence. This drives the first few feature detectors to shortly respond to the onset transition. Next, all the 3 syllables exhibit a subsequent flat harmonic stack, which turns into sustained responses of a large group of detectors. These detectors are selective for these harmonic stacks and will respond for as long as their preferred pitch range is matched. Finally, for the first and last syllables, the harmonic sweeps are converted into sequences of brief activations. When we apply $F$ to the responses in Fig. 3.6B, we obtain a very sparse sequence of spikes.
3.2.2 Training

Now that we have discussed the concept of detectors for short-time vocal events and how to encode syllables as sequences of detections, we will look into the implementation. The first problem we have to solve is grouping feature vectors. As we mentioned, we will assume that a representative syllable for each type has been carefully selected prior to training.

3.2.2.1 Selection of acoustic events

In principle, we could apply DTW to every pair of syllables and then cluster the feature vectors based on the DTW alignments. For example, suppose we construct DTW alignments for 3 syllables in the same category. The unions of the 3 alignments can be interpreted as the edges of a graph where the vertices are represented by feature vectors. In other words, if \((x_{\psi(k)}, y_{\psi_x(k)})\) represents the DTW alignment between syllables \(x\) and \(y\), the feature vectors \(x_{\psi_x(k)}\) and \(y_{\psi_x(k)}\) will connect by an edge. Next, we can perform spectral clustering to find groups of strongly connected vertices. The groups will represent the clusters we seek. This approach has been effectively used for unsupervised lexical discovery in continuous speech [67]. However, computing DTW alignments for every pair of syllables is computationally expensive. Since we’re not seeking to generalize...
across speakers, as is normally done in ASR, and since Bengalese finch syllables of the same type show little acoustic variations, we should be able to afford a simplified method.

Recall that the level of DTW distortion is small for Bengalese finch syllables. For this reason, we’ll adopt a greedy approach. First, to avoid comparing every pair of syllables, we pick one exemplar per category to serve as the representative. Next, we remove the DTW constraints and perform a search for the feature vectors which are the closest to each of the reference syllable’s feature vectors. Our assumption that the feature space is normalized to the surface of the unit sphere comes in handy now. The cosine distance between the feature vectors can be computed by cross-correlation, which can be efficiently implemented with FFT.

Fig. 3.7 compares the cross-correlation method versus DTW for Bengalese finch syllables. Differences are minor, including for noisy and broadband syllables. This justifies using the maximum of the cross-correlation for efficiency. However, for more complex vocalizations or speech, this is likely not going to hold. In a speaker independent system, for example, additional DTW constraints might be necessary in addition to speaker normalization.

We can increase the selectivity of clusters by imposing a threshold to the maximum cross-correlation value. Feature vectors with a maximum cross-correlation below this threshold are removed from the clusters. In practice, we set it to 0.5. In effect, this tightens the clusters around their seed vectors and help prevent poorly annotated syllables.

Figure 3.7. Clustering short-time events in syllables. Left: using the peak in the cross-correlation between pairs of syllables. Right: using the DTW alignment between pairs of syllables.
from distorting the decision boundaries. This is particularly relevant for hard-margin SVMs, where a single support vector can completely distort the decision boundary. Note that when this threshold is set too high, real positive vectors may be tossed away and cause the performance to degrade.

This covers the collection and clustering of similar acoustic events. In principle, we could construct decision boundaries such as in Fig. 3.4B using the positive vectors only. Conceptually, we would like to measure location and scale of these clusters to estimate the support of the density of positive vectors. In practice, we could use one-class SVMs, where the support vectors are selected by maximizing the margin against the origin. A more principled way is to take into account the existence of negative feature vectors.

Ideally we should use the entire set of negative syllables to construct the set of negative vectors. But recall that syllables may overlap. Fig. 3.8A illustrates this with two Bengalese finch syllables. The two syllables on the top share a common start of a few milliseconds. The two in the bottom are composed of a shared subunit followed by distinct parts. Any attempt to construct a decision boundary that separates these overlapping portions will be subjected to overfitting. One way to prevent this problem is to ignore some of the overlapping portions. Fig. 3.8B illustrates this idea. On the top, the decision boundary avoids the mid negative feature vector by splitting itself in two, leading to a complicated decision boundary. In the bottom, the mid vector is ignored and the boundary becomes much simpler. To implement this idea, we set a threshold for the angle between each negative vector to each positive vector, as illustrated in Fig. 3.8C. If the threshold is crossed, we remove the negative vector from the negative set. This threshold also needs to be carefully selected, since it may affect each syllable type differently. A large threshold may remove relevant negative vectors which help separate two distinct acoustic events, leading to decision boundaries that are too simple and can’t discriminate properly.

We also need to address how detectors behave in the presence of silence and noise. Ideally they should remain unresponsive. But noise is an umbrella term. It includes everything that isn’t part of the song and isn’t silence either. How do we prevent detectors from responding to background sounds such as wing flaps, footsteps, and calls? Many classes of sounds are unpredictable, e.g. the barking of a dog. The set of feature vectors generated by the song covers only a small fraction of the feature space. It’s unlikely that our decision boundaries will automatically prevent detectors from responding to unseen categories. Calls, for example, may sometimes resemble syllables. It’s possible that an unseen call will explore some region of the feature space where the syllables alone weren’t
able to constraint the decision boundary. This will cause the detector to respond to the call. To prevent insertions like this, we collect feature vectors representing silence and noise and include them in each negative training set. From each recording, we first collect at most 30 contiguous segments between 80 ms and 300 ms which have not been annotated as syllables. Typically, for lab recordings, these segments will mostly contain silence. Silence is whatever stationary sounds are present in the background throughout the recording. So, for each detector, we randomly sample about 5 s (of the order of 1,000 vectors) from the segments and add it to the negative set.

While silence is stationary and can be defined by its statistics, calls, wing flaps, and footsteps are not. It is likely that this last step will miss brief but loud noises. So we also need to ensure that brief background noises which are not syllables are also sampled. In perfect recording settings, this is not needed. We observe that feature detectors remain

**Figure 3.8.** Preventing complex decision boundaries when syllables share features. (A) Two pairs of Bengalese syllable types that share features. Top: the two syllables share the same start. The following portion in one syllable type is always a flat harmonic stack, while the other is an upward sweep. (B) Top: adding shared feature vectors in both positive and negative sets leads to complex decision boundaries. Bottom: removing the shared vectors from the negative set simplifies the boundary. (C) Deleting vectors from the negative set lying too close to the positive seed vector. Negative vectors lying within the cone are pruned out. (D) The same template when units are trained using various thresholds, \( \cos(\Omega_{\text{min}}) = 0.5, 0.7, 0.9, 0.95, 0.98, 1.00 \). Only the indicator functions over the sets \( z_i > 0 \) are shown.
unresponsive in the absence of noise. However, both the strong compression and the normalization scheme we use in feature processing (see Sec. 1.2.1.2) make our detectors more susceptible to brief noises. Such noises can drive insertions in the output and mask the performance of detectors during syllable recognition. In the results we show in Sec. 3.3.7 of this chapter, we circumvented this problem by calculating the recognition error for portions of the recordings between the first introductory note and the last syllable. However, for the following chapters, we included the entirety of every recording. To prevent insertions, we randomly sampled contiguous chunks spanning about 10,000 frames (roughly 30 s, when available) and of at least 2 standard deviations above the average power of the previously collected silence. Here we define power as the amplitude of the short-time energy profile of the waveform. Differently from the negative vectors from syllables, we do not prune out silence or noise based on the angle they make with the positive samples.

3.2.2.2 Template training

Templates are stored pattern which can be used for matching input patterns. Matching requires both template and input to be in the same representation. To construct our templates, we feed the detectors with the same syllables we used for training. We then pool and store the responses. By design, most detectors only respond to their positive syllable type (e.g. Fig. 3.6B). Also recall that the SVM margin guarantees that detectors will respond to at least one time point for every positive vector $x_i^+$. But in cases where syllables share features, it is still possible for detectors to respond to more than one syllable type, especially when $\Omega_{\text{min}}$ is small. To control pooling, we use thresholds for the average amplitude of responses and the rates of responses. A minimum average amplitude prevents pooling detectors that respond weakly. Similarly, a minimum rate of response for a syllable type prevents the addition of unreliable detectors. These two thresholds increase sensitivity. A minimum rate discards detectors with low specificity. In practice, we observe that sharing amongst syllable types is rare for SVM detectors. Since a template accounts for a fraction of the acoustics of a syllable type, we store multiple templates per type.

While it is not necessarily true that the seed feature vector will be selected as a support vector during SVM training, it frequently is. Therefore, we force the order in which seed features occur in the seed syllable onto the templates of the same syllable type. This results, in most cases, in non-decreasing sequences of spike times.

In Fig. 3.9A we illustrate how differences in the acoustics lead to variations in the
Figure 3.9. Relative delays in templates of syllables. (A) 3 Bengalese finch templates of the same syllable type. Top: relative spike time matrices. Middle, bottom: spike templates. (B) Relative delay distributions of 7 units in 30 templates of the same syllable type (same syllable as in A). Note the broadening of the distributions as the units become further distant in the sequence.

spike patterns in the templates. The top row shows the normalized relative delays between spikes in each sequence. Panel 3.6B shows the statistics of relative spike times between 7 units and 30 templates of the same syllable type. The larger the expected delay between units, the larger the variance tends to be. But we also observe that the distribution of relative delays between rapid acoustic transitions tend to be narrower. Conversely, flat harmonic stacks tend to be broader and sometimes multimodal.

### 3.3 Continuous recognition of ultra-sparse sequences

In this section we describe our attempts at recognizing syllables from spike sequences using spike templates. In the end of the section we report results for continuous recognition of Bengalese finch syllables when training data is scarce.
3.3.1 Sequence detection with delay lines

When the input stream and the template contain the same sequence, we can easily perform decoding with delay lines. A delay line simply compensates for the expected relative delay between two units so that their spikes synchronize in time. At the end of each delay line we place an integrator which sums spikes within a fixed temporal window. Whenever the input spikes arrive together within this window, the integrator’s internal state triggers an event. This can be accomplished minimally with a leaky integrate-and-fire neuron (LIF). In the LIF neuron, incoming excitatory spikes evoke an excitatory postsynaptic potential (EPSP) which depolarizes the neuron’s membrane potential up to a fixed threshold $\theta$. Once this threshold is reached, the LIF neuron spikes and subsequently returns to a resting state. The LIF’s subthreshold dynamics is described by an RC circuit, where the time constant represents the integration window.

To understand how the LIF neuron can help us decode the input stream, suppose $N$ spikes arrive at the neuron’s terminals at times $t_1, \ldots, t_N$ through the delay lines. Each spike contributes with a certain strength $w_i$, $i = 1, \ldots, N$. The specific value of $w_i$ depends on the properties of the synaptic terminal $i$ of the LIF. The total input current is given by $I(t) = \sum_i w_i \delta(t - t_i)$, and the membrane potential $V(t)$ integrates the current in time:

$$V(t) = \int K_\tau(t - t') I(t')dt' = \sum_{i=1}^N w_i K_\tau(t - t_i) .$$

(3.6)

In the equation above, $K_\tau$ is the exponential kernel $K_\tau(t) = \exp\{-t/\tau\}$ with time constant $\tau$. The LIF’s subthreshold potential is a linear filter with impulse response $K_\tau$. When most spike times coincide, $V$ grows fast and drives the neuron to spike. This spike represents the detection of a stimulus matching the delay times in the delay line. Fig. 3.10A illustrates the LIF mechanism coupled with delay lines.

To recognize different combinations of delays from the same set of input units, we can extend the delay lines to a delay grid. Fig. 3.10B illustrates a delay grid that selects for two sequences. The input is shown to the left as three simultaneous spikes. When sent through the delay grid, the output in the bottom becomes a temporally stretched version of the one on the top. In the context of sound localization, this concept of coincidence detection using relative delays in a spatial grid is known as the Jeffress Model.

For real acoustic data, the input sequence and the templates will rarely be a perfect match. Implementing one delay line per combination of sequence delay is not realistic. We can take advantage of tuning the connection weights, threshold potential, and the time constant to help the delay grid recognize more temporally variable sequences. For
Figure 3.10. Delay grid of LIF neurons for temporal pattern matching. (A) Left: A LIF neuron can be selective for specific temporal sequences when conduction delays are imposed before the connections. When units 1, 2, and 3 fire simultaneously, unit 4 does not spike (left). When the relative delays of their spikes compensate for the conduction delays, unit 4 spikes. (B) A delay grid can select for multiple temporal sequences by pooling the delay lines at multiple points. In this illustration, conduction delays increase along the horizontal axis.

example, increasing the time constant widens the integration window and allows for a larger temporal warping. However, more difficulties emerge when we attempt to fine tune this. First, if the threshold is fixed, the tuning of one connection weight needs to be traded off by the tuning of the rest of the weights and the kernel’s time constant. This is a difficult credit assignment problem. Second, there is also a trade-off between how much temporal warping we allow in the set of true positive spikes and how many false positives will trigger a detection in the end. Third, to perform recognition in auditory scenes, we must be able to reconstruct the sequence of acoustic events that gave rise to an auditory object, not just detect the presence of the auditory object. In the LIF circuit of Fig. 3.10A this would correspond to the inverse problem of recovering the input spikes from the LIF’s output spike.

Instead of directly addressing the first difficulty by optimization, e.g. as done in [68], we’ll try to circumvent it. Recall from Fig. 3.6 that we construct a spike representation by taking the peaks of the response traces above a fixed threshold. In particular, the response clipped at the threshold ($z_i(t)$ in Eq. 3.4) in practice vaguely resemble the weighted EPSPs $w_iK_r(t)$. We’ll take an inductive leap and use $w_iK_r(t - t_i) = z_i(t - t_i)$. This is, of course, not a LIF neuron anymore, but the analogy is conceptually helpful. In any case, the membrane potential of the integrator unit can be denoted as

$$V(t) = \sum_i z_i(t - t_i), \quad (3.7)$$
Figure 3.11. Delay grid of feature detectors for template matching. (A) The template is aligned to the input stream by the first spike (first vertical dashed line). The following spikes may show some degree of temporal distortion (dashed lines). (B) The relative spike times in the template define a delay line which we use to align and integrate the input stream. The alignment quality $w$ is large when the delay line correctly compensates the relative spike times in the input. (C) A high alignment quality $w$ suggests a true positive. The sequence can be recovered by searching for synchronous spikes underlying $w$.

where the summation index now runs over a subset of the feature detectors.

3.3.2 Formation of auditory objects using a simple delay grid

Fig. 3.11A illustrates an input stream of spikes matched against a template. The first spike is used as a reference to temporally fix the template relative to the input. The leftmost vertical dashed line represents this alignment. We can use the time interval between the reference and the subsequent template spikes to shift and align the input spikes exactly as a delay line would. The relative delays in a template define a delay line and a set of templates define a delay grid. Fig. 3.11B illustrates the delay line. The traces in the bottom show the membrane potential at the output unit with the assumptions discussed in the previous section. Equivalently we can also think in terms of pushing back in time the later spikes, as shown in Fig. 3.11C.

Another intuition for $z_i(t)$ at the moment of spike is given by Fig. 3.5B. The height of $z_i$ above the baseline reflects how deep into the decision boundary the acoustic event goes. We interpret the height of the peak in $z_i$ as the quality of detection. Similarly, we interpret the height of the peak in $\sum_i z_i(t - \tau_i)$ as the quality of the sequence alignment.
In the absence of insertions, this alignment quality is:

\[ w = \max_t \sum_k z_k(t - t_k) \, . \] (3.8)

Now assume that \( w \) is large enough to give us confidence that it really underlies the correct detection of a sequence. This gives us a robust temporal landmark to start collecting the spike sequence. We’ll call the collected sequence an auditory object, in analogy with auditory scene analysis. This is represented in Fig. 3.11C by the boxed spikes. Template matching thus binds short acoustic events together into sequences to generate auditory objects. In the following sections we will look into two mechanisms for generating auditory objects. We will first use the width of the flanks surrounding the peak of height \( w \) to locate and bind the spikes under it. The difficulty with this method is that it depends on a well tuned width or threshold parameter. In addition, it may not work well for sequences of variable duration. To circumvent these difficulties, we’ll introduce a method based on graph search for sequence alignment.

### 3.3.3 Balancing insertions and deletions

In the previous section we hinted at the use of the membrane potential \( V(t) \) at the peak position as a reference for searching other spikes in the sequence. This strategy can work well under small temporal warping. As the warping factor increases, the height of the expected peak in \( V(t) \) decreases, lowering the alignment quality \( w \). The threshold for \( w \) may be lowered to counteract this effect, but only up to the point when insertions start being detected. But even when the amount of warping is negligible, this idea can fail in practice when the detectors are not robust to insertions. For instance, it may overshoot \( w \) when there are inserted spikes. It can also be sensitive to deletions if the threshold on \( w \) is high. Another issue is that it is unable to account for multiple spikes from the same unit within a single sequence.

The issues with insertions, deletions and threshold tuning can be understood with the following toy problem under the assumption of Gaussian subthreshold responses. That is, \( z_k(t) = 1/N \sum_i \exp\{-0.5(t - t_i)^2/\sigma_S^2\} \), where the sum extends to all the \( N \) spike times \( t_i \). This is equivalent to a kernelized spike train. We add Gaussian jitter of width \( \sigma_J \) to spike times in the test sequences to simulate differences between the template and the inputs. The fraction \( r = \sigma_J/\sigma_S \) controls the noise-to-signal ratio. At low values of \( r \), most test sequences should align nearly perfectly with the template and the alignment quality \( w \), i.e. the maximum of \( V(t) \), should be close to 1. At larger values of \( r \), the
alignment will degrade and \( w \) will decrease (Fig. 3.12A). The statistics of the alignment quality starts narrow around 1 for low \( r \). For larger \( r \), it broadens and shifts towards the origin (Fig. 3.12B).

The generalization error for a single delay line represents the chance of detection above the threshold. In the absence of insertions, deletions and other competing templates, this error is easy to interpret. It corresponds to the fraction of the distribution of alignment qualities \( w \) below the threshold. Fig. 3.12C shows the generalization error for multiple noise-to-signal conditions and sequence lengths. For longer sequences, the distributions narrow, forcing the errors to increase faster and plateau sooner.

The presence of deletions or insertions in the spike code cause the error curves to shift left or right respectively. Fig. 3.13 shows the error when deletions or insertions have a fixed non-zero probability of 20%.

At this point, the problem is trivial since we can always fix a threshold low enough so that detection errors are completely absent. Continuous recognition introduces a major step-up in difficulty because external sources may generate false positives. In birdsong recordings, for example, pecking and wing flaps are inevitable sources of background noise. To understand how false positives affect the generalization error, let us assume a common source of false positives. Call \( D \in \{0, 1\} \) the random variable associated with the delay line’s detection (1 for detection, else 0), \( T \) the main source, and \( N \) the background source. The generalization error can then be written as

\[
\epsilon = P(N)P(D = 1|N) + P(T)P(D = 0|T),
\]

where we have assumed that \( P(N) + P(T) = 1 \). The problem is suddenly not trivial. We may have false positives and false negatives as sources of errors. As Fig. 3.14 shows, a fixed threshold for the delay line will not solve the problem for all conditions. To deal with this issue we will develop new ideas in the next sections to help separate the distribution of target and background sources.

### 3.3.4 Sequence matching using delay distributions

So far our delay lines make use of the relative delays from a single template. Can we use the statistics of relative spike times (see Fig. 3.9B) to increase the scores of true positives? Let \( s_i = (i, t_i) \) denote a spike from unit \( i \) at time \( t_i \). Estimating probabilities of the form \( P(s_i|s_{i-1}, \ldots, s_1) \) is difficult because of our assumption of small training sets. But we can calculate bigrams of the form \( P(s_i|s_j) \) using the spikes in the templates. In particular,
Figure 3.12. Generalization error using delay lines for recognition. Responses of units are Gaussian bumps of width $\sigma_S$. Spikes times are assigned to the center of each Gaussian. When perfectly aligned by the template’s relative delays, the sum of the bumps, $V(t)$, is a Gaussian with height 1. The test sequences are copies of the templates with added Gaussian jitter of width $\sigma_J$. (A) As the noise-to-signal ratio $\sigma_J/\sigma_S$ increases, the sequences become more jittered and the maximum amplitude of $V(t)$ decreases. (B) The distribution of alignment quality, $\max_t V(t)$, broadens and shifts towards lower values as the SNR increases. Here shown for sequences with a total of $|S| = 10$ units. (C) Generalization errors for the detection task with a fixed threshold value. Detection occurs when $V(t)$ crosses the threshold only. Increases in the SNR drive the error curves to plateau sooner. Sequence length, $|S|$, determines how fast the errors plateau.

we can estimate $P(s_i|s_{i-1})$ whenever both $s_i$ and $s_{i-1}$ exist. Given the first spike $s_1$, we can use these bigrams to approximate the probability of sequence $S = (s_1, s_2, \ldots, s_N)$ to match with this set of templates using the Markov assumption,

$$P(S|s_1) \approx \prod_{i=2}^{N} P(s_i|s_{i-1}).$$

(3.10)
Figure 3.13. The effects of insertions and deletions in the generalization error. Errors when uniform probabilities of insertion $P_{\text{ins}} = 0.2$ and deletion $P_{\text{del}} = 0.2$ in the spike sequence are taken into account for 3 SNR conditions. Insertions displace the error curves to the right, while deletions displace errors to the left. Higher SNRs displace all curves left.

This equation represents a chain starting at $s_1$. An even simpler idea is to assume independence given the first spike in the sequence. For example,

$$P(S|s_1) \approx \prod_{i=2}^{N} P(s_i|s_1). \quad (3.11)$$

In our early attempts, we tried approximating the bigrams by Gaussians and estimating the mean and variance using maximum likelihood. However, this turned out to yield poor estimates, particularly when only 5 to 10 syllables per syllable type were available for training. The bigrams in practice can be multimodal and hard to estimate. Fig. 3.15A illustrates why. Our spike representation assigns spike times at the peaks of the
detector’s responses. For tonal syllables, small fluctuations can change the spike time drastically.

Fig. 3.15B shows 5 histograms of relative spike times for the same Bengalese finch syllable when more than 20 syllables are available for training. To use these histograms, we first need to smooth them. Equivalently, we can use a Parzen window to estimate the delay densities,

$$\hat{p}(t) = \frac{1}{Nh} \sum_{i=1}^{N} K \left( \frac{t_i - t}{h} \right).$$  \hspace{1cm} (3.12)

In this equation, $N$ is the sample size, $K$ is a kernel, $h$ is the bandwidth and $t_i$ are the relative delays. For the smoothed histogram shown in blue in Fig. 3.15B we used a Gaussian kernel, but in the following panels the results are for a Laplace kernel, as it tends to be more robust to outliers. The main challenge is selecting the bandwidth $h$ using a few observations. A popular criterion is to analyze the mean squared error between the estimated density $\hat{p}(t)$ and the histogram. This is justified by a trade-off between the bias and the variance terms of the error. When $h$ is small, the variance is large, but the bias is small, vice-verse when $h$ is large. But since the histogram is not the actual distribution, this turned out to be a difficult approach.

The assumption that the first spike triggers detection may lead to an increase in the number of sequence candidates and generate insertions. To circumvent this, we set a small threshold to the minimum probability that a sequence candidate has to accumulate. Another difficulty with this approach comes from deletions. Here a deletion will necessarily zero out the expression, so we need to deal with a missing value. The simplest idea is to guess a probability value for missing values, known as an imputation. Alternatively, we can just ignore the missing value.

We tried using the delay distributions in two ways for decoding. First, we tried generating a continuous cost trace based on the independence assumption of Eq. 3.11 for each syllable type. This was simply the negative logarithm of Eq. 3.11 plus a constant. Fig. 3.15C shows an excerpt of a Bengalese finch song along with the spikes for the same syllable type. The regions highlighted in red show the widths of Gaussian fits to the bigrams $P(s_i|s_1)$, with $i = 2 \ldots N$. In the third panel we show the cost trace. We assign the minimum value of the cost throughout the entire duration of the sequence matched. We set the imputation value to $1e-2$. Note in particular how the independence assumption fails to adjust for the beginning of the sequence.

Next, we used the Markovian assumption of Eq. 3.10. However, for the missing values, we used a backoff approach. For example, if spike $i$ was missing from the sequence,
Figure 3.15. Distributions of relative delay. (A) Spike sequences for 3 Bengalese finch syllables. Assigning spike times to the peaks of the responses creates sequences vulnerable to small acoustic variations. (B) Bigrams of spike times for the syllable in panel A. The histograms show the spike times of detectors when the spike time of the previous detector in the sequence is given. The blue curve shows the histogram after smoothing with a Gaussian kernel. (C) Independence assumption of spikes during decoding with a cost function. Bigrams from panel B were fit by Gaussians. The regions highlighted in red show 1 standard deviation for the bigrams $P(s_i | s_1)$. (D) Chain assumption during decoding. The regions highlighted in red show 1 standard deviation for bigrams $P(s_i | s_{i-1})$ fitted with a Laplacian kernel.

we took $P(s_{i+1} | s_i) \approx P(s_{i+1} | s_{i-1})$. The progression of the chain for the same Bengalese finch syllable is illustrated in Fig. 3.15D.

What is particularly interesting about the bigrams is not necessarily their specific values, but that they roughly indicate the expected time interval for spikes in the sequence. We found it difficult to fit the delay histograms in the regime of small training sets, so we didn’t push the cost function idea further. But the bigrams can be used as strong priors in the following way. Take a fixed probability value and define a characteristic
function to give 1 where the bigram is larger than that value and 0 everywhere else. This characteristic function defines a small interval where we can search for the spike. To avoid the trouble of estimating the bandwidth for the kernel density estimator, in the next sections we will go one step further and interpret the detector’s response $z_i(t)$ itself as a proxy for the delay distribution.

### 3.3.5 Search-based sequence reconstruction

In this section we describe a method for reconstructing spike sequences using stored templates.

#### 3.3.5.1 Spike-alignment graph

As in the previous sections, we assume that both template and input have been generated by the same ordered collection of detectors. Spike times are assigned to the peaks of the EPSPs. To fix the notation, we represent the first spike in the input by $s_1 = (d_1, t_s^1)$, where $d_1$ is the unit’s id and $t_s$ the spike time. We represent the first spike in the template by $u_1 = (d_1, t_u^1)$. To keep the notation simple, we avoid indexing spikes when there is degeneracy, that is, when a unit spikes twice or more. We also index the response traces $z^s_i(t)$ and $z^u_i(t)$ to indicate whether it corresponds to the input or the template.

We call the tuple $a_i = (s_i, u_i)$ an alignment node and any set of the form $A = \{a_1, \ldots, a_N\}$, with no repeated entries with the same unit index, an alignment. Our goal in this section is to search for an alignment that allows for the recovery of the most important spikes in the input. For that, let us assume that the first alignment node $a_1$ is given. In Fig. 3.16A, this corresponds to the red vertical line. With $a_1$ fixed, the next unit’s EPSP trace indicates the time interval when the next spike, $s_2$, is expected.

Previously we saw that estimating the density $P(s_i|s_{i-1})$ from a few observations can be difficult. Instead, we will loosen the restrictions on template matching and define an indicator function $\xi_i$ that takes 1 when $z^u_i(t) > 0$ and 0 otherwise:

$$\xi_i = 1 (z^u_i(t) > 0), \quad (3.13)$$

where $1$ is the indicator function. This way, $\xi_i$ narrows down the search interval where the spike $s_i$ is expected to be found. Fig. 3.16A illustrates the search interval, the non-zero support of $\xi_i$, as a red box. Together, the alignment node $a_1$ and $\xi_2$ help prevent insertions by narrowing down the search space (first panel, Fig 3.16B). The source of
deletions can also be identified. Finally, units expected to fire twice or more can also be recovered (third panel, Fig. 3.16B).

Once $s_2$ has been retrieved with the help of $\xi_2$, we create a new alignment node $a_2 = (s_2, u_2)$ and continue the search. Since we don’t have the concept of a state here, we don’t have a trellis to perform the search over, as typically done for decoding in ASR. Instead, we build an alignment graph on the fly. Nodes in the graph represent alignment nodes and transitions indicate the sequence of selected alignment nodes (Fig. 3.16C). An end node is attached to the last node once the last unit in the sequence is reached. We also attach a start node transitioning into the first alignment node to represent the start of the graph.

The alignment graph shown in Fig. 3.17A illustrates the construction in the presence of an insertion. At the first spike, $s_1$, the node $a_1$ is connected to the start node. During the search, $s_2^{(1)}$ and $s_2^{(2)}$ are found, so $a_1$ transits into two nodes, $a_2^{(1)}$ and $a_2^{(2)}$. The next spike, $s_3$, is only visible from $a_2^{(2)}$, so the alignment node $a_3$ is only connected through $a_2^{(2)}$. All the leaf nodes are connected to the end node and this concludes construction of the graph. The question now is how do we select the best alignment from the alignment graph?

In the example of Fig. 3.17A, the longest path in the directed and unweighted graph $A$ represents the best alignment. Starting from the end node, we use backward tracing to determine the longest path. Note that since $A$ is a directed acyclic graph (DAG), the
longest path can be efficiently calculated in linear time. The longest path gives us the maximum number of active units that can be reconstructed using the template. This is sufficient when the template contains no units spiking twice or more. We set the final matching score to $|P_A|/|T|$, where $|P_A|$ is the length of the longest path in $A$ and $|T|$ is the total number of spikes in template $T$.

This idea helps to deal with insertions, deletions, and in specific situations such as when a unit is expected to spike twice. However, as Fig. 3.17B illustrates, it fails in more complex cases. The two alignments in Fig. 3.17B have the same length, but only the left one allows the recovery of all spikes. In order to extend it, we need to assign weights to the transitions.

The weighting scheme has to be such that it breaks the degeneracy between the two cases of Fig. 3.17B. The best node at each step should allow for the recovery of the maximum number of spikes. So we weight the outgoing transitions by the total weight of the spikes that the parent node recovers. For example, node $a_1$ in Fig. 3.17C makes two spikes visible, one with associated weight $w_1$ and the other $w_2$. Thus, every transition from $a_1$ is assigned a total weight equal to $w_1 + w_2$. Note that this weighting scheme resolves Fig. 3.17B, but Fig. 3.17C is still ambiguous. We will ignore this type of ambiguity for now and revisit it in the next chapter.

The matching score also needs to be modified. Assuming the weights are at most 1 and that $|T|$ is the number of spikes in the template, we set the matching score to

$$
\min \left\{ \frac{1}{|T|} \sum_{e \in P_A} w_e, 1 \right\},
$$

where the sum is over the weights visited by path $P_A$ in the alignment graph $A$. The min operation ensures the score is normalized even in cases such as the one shown in Fig. 3.17C.

Note that we construct and traverse one alignment graph per template, which can be a lengthy calculation overall, particularly when the number of templates and units are large. However, as the templates are independent of each other, multiple template matching routines can be carried concurrently. In addition, as Fig. 3.17B illustrates, many of the alignment nodes may not be necessary to consider. In fact, we observe that insertions typically consist of small bumps above the baseline. Therefore, we use beam search with a small beam size ($\text{beam} = 2$) to prevent long runtimes. We sort the beam by weight $w_i$ to ensure the most important nodes are expanded first.

One difficulty with this approach is that it attempts to construct a new graph every
Figure 3.17. Weighted spike alignment graphs. (A) Even in the presence of an insertion, this particular unweighted graph has a unique longest path. (B) In more complex cases, the solutions may be degenerate while recovering different numbers of spikes. The left alignment allows for the recovery of the entire sequence, while the right alignment doesn’t. (C) Weighting the edges of the alignment graph helps to break this degeneracy in some cases. Here we assign the sum of the heights of the subthreshold trace at the spike times as the weight.

time it encounters the first spike of a sequence. In practice, because we may also have insertions, this would be inefficient. To reduce the number of candidates, we use delay lines in the regime of low thresholds, which ensures fewer deletions. In this case, when delay lines respond, it’s an indication of either an insertion or a true positive. We then proceed to construct an alignment graph to carefully assess which is the case. We will go into more detail about this idea in Sec. 3.3.5.3.

3.3.5.2 Sequence completion

With the help of the alignment graph, sequence completion is straightforward. If a spike is missing, we select an alignment node to fix the search interval and then proceed to iteratively lower the missing unit’s threshold (Fig. 3.18A). We lower the threshold in small steps to avoid introducing errors from subthreshold fluctuations. Once the spike is found (or the threshold hits the baseline) the threshold is restored to its original value.

The same process is repeated for every missing unit, but the choice of the alignment node needs to be adapted. There is always a chance that the selected node may not provide an effective search interval. This may happen, for example, when the alignment node’s unit and the missing unit are on opposite ends of the sequence. Recall that even in the absence of temporal warping, the conditional distributions $P(s_i|s_j)$ tend to broaden the more separated units $i$ and $j$ are in the sequence (see Fig. 3.9B). To reduce this risk, we adopt some heuristic choices for the alignment node during sequence completion. Let the missing unit be the $i$-th unit of the sequence. If $i$ is the start of the sequence, select
Figure 3.18. Sequence completion with an alignment node. (A) Given an alignment node and a missing spike, we can lower the threshold of the missing unit and use the non-zero support in the template to search for it. (B) Sequence completion in a Bengalese finch syllable. Left: before sequence completion. Right: after completion, with spikes found in red. (C) WER for a Bengalese finch (s933) with and without sequence completion. Sequence completion reduces the WER by about 1%.

Fig. 3.18B shows sequence completion for a Bengalese finch syllable. Recovered spikes are highlighted in red. Panel 3.18C shows a benchmark of continuous recognition of syllables from an adult Bengalese finch (age unspecified). In total 25,845 syllables in 18 syllable types were recorded and manually labeled for this bird, including introductory notes. Roughly 37% of these were used to compute the error rates in panel 3.18C. Training exemplars were selected at random from the remaining 63% of syllables. Each training condition (horizontal axis in Fig. 3.18C) adds 5 new training exemplar per syllable type, which corresponds to a total of $5 \times 18 = 90$ manually annotated exemplars. Although introductory notes were used for both training and testing, they were removed from the manual annotations and output string prior to calculating the WER. In the next chapter we will include introductory notes in our benchmarks.

Panel 3.18C highlights a decrease of about 1% in the WER when sequence completion is turned on. This error difference seems more pronounced for smaller training sets, suggesting that spike deletions are more likely in that regime. Intuitively, the smaller the training set, the larger the margins tend to be in the hard-margin SVM. So we suspect that this error difference comes from our implementation of sequence completion, where we search for spikes missed within the region between the decision boundary and negative margin.

Now, sequence completion may help improve the quality of the reconstruction of a
true positive, but it may also do the same to false positives. Additionally, we used the delay grid in the regime of low thresholds to detect syllable candidates, which induces the appearance of insertions. Thus, we need an additional mechanism to eliminate insertions.

### 3.3.5.3 The prediction stream

Template matching generates a stream of predictions. These predictions bind spikes from the input stream together into sequences that represent chunks of audio. So, for our purposes, a prediction is a collection of spikes with a semantic label and the score of Eq. 3.14 attached to it. From the spike sequence we can also determine the instant when the prediction was generated and its duration. But not all prediction are useful. The songs we’re analyzing are composed of sequences of syllables separated by silent gaps and never coarticulate. So it’s reasonable to require a unique prediction for each syllable in the song. To proceed with recognition, we have to eliminate all the insertions we generated during template matching and leave only a single prediction per syllable. For that, we need to distinguish between the following:

1. False alarms and true positives;
2. True positives of the same syllable type;
3. True positives of different syllable types.

The first distinction refers to insertions due noise or other vocalizations which are not syllables. Recall that, in addition to directed songs, songbirds may also produce numerous other vocalizations to communicate specific behaviors. The second distinction is important because templates of the same syllable type may bind different sequences. As a consequence, the boundaries of the predictions may also be different. This is typically not the case for Bengalese finches, as syllables are stereotyped. For Bengalese finches, it’s sufficient to select the match with the largest fraction of recovered spikes in respect to the template. However, we’ll see in the next chapter that this may not work well for the canary. The third distinction reflects the fact that syllables may share acoustic features, and, as we’ve seen in Sec. 3.2.2.1, may sometimes even share entire subunits. So how do we get rid of all but the correct prediction? To fix notation, we’ll assume that a prediction in the interval 

\[ [t_1, t_2] \]

overlaps with another in 

\[ [t_3, t_4] \]

if

\[ |[t_1, t_2] \cap [t_3, t_4]| > \alpha \min (t_3 - t_4, t_2 - t_1) \]

(3.15)

In this equation, \( \alpha \) is the overlap fraction. We fix \( \alpha = 0.5 \) in our benchmarks.
During recognition, we compare predictions pairwise and store all the overlapping pairs in a buffer until we encounter an interval containing no predictions. For example, if syllables $A$, $B$, and $C$ overlap, the queue will contain the sets $A, B$, $A, C$, and $B, C$. At this point, the decision module is triggered. It first ranks all the buffered pairs in a queue by their absolute differences in score. Pairs with large score gaps are readily ignored for efficiency. When a syllable is removed from the queue, all the pairs where it appears are removed as well. Pairs with a small score gap require a more careful analysis, particularly when sequence completion is used. We fix this score margin at 25%. We suggest three ways to compare pairs with a score margin under this value:

(a) Compare simple acoustic features (e.g. pitch) of the input with the templates;

(b) Use the alignment graph to compare the acoustics of short segments of input with the templates;

(c) Use specialized feature detectors to discriminate between two syllable types.

In this chapter, we look into (a). We first compare the power spectrum of the song interval within the prediction’s interval to the average power spectrum of the templates in the prediction’s class. After this step, if there are still unresolved entries in the queue, we use the prediction’s score. However, we note that duration may be a better predictor for Bengalese finch syllables. But since our goal is to extend the system to more songbird species, we decided to not press further with this assumption. We discuss items (b) and (c) in the following chapters.

In Sec. 3.3.5.1 we mentioned the use of delay grids prior to sequence reconstruction for fast detection. Since templates for Bengalese finch syllables are typically similar, slight variations in the amplitude of $V(t)$ can be washed out by averaging. For a syllable type with $N_T$ templates and $N_d$ feature detectors, we use the average

$$\langle V(t) \rangle = \frac{1}{2N_T N_d} \sum_{k=1}^{N_T} V_k(t)$$

(3.16)

to detect new candidates. The factor of $1/2$ appears here to normalize $V_k(t)$, since it typically lies within $[0,2]$ (in the SVM formulation, we rescale the functional margin to 1, so the total span between the two margins is 2). We fix a threshold $V_0 = 40\%$ for $\langle V(t) \rangle$ representing the minimum average alignment quality required for detection. Detection then triggers sequence reconstruction for all templates of that same syllable type starting from the earliest spike. Templates within each syllable type compete with one another
with the scores of Eq. 3.14. Thus, at each detection step, only a single prediction is sent to the prediction stream by a set of templates. In the next chapter we will soften these assumptions.

Our discussion so far covers distinctions (2) and (3). Addressing (1) is difficult when we don’t have examples of false alarms. When we do, for example we have a few alarm calls which we identified when manually annotating the songs, we can use the strategy in item (a). Conversely, we can collect and use noisy chunks from the recordings. Otherwise, we are forced to require all output predictions to have a minimum score and drop predictions below this minimum. This may lead to deletions, and we will need to address this possibility in the following chapters.

### 3.3.6 Feature manipulation

Before we benchmark our ideas, we briefly discuss transformations in the feature space. Can we improve the system’s performance by manipulating feature vectors? In this section we summarize two of our attempts at feature augmentation.

#### 3.3.6.1 Spectral entropy

Spectral entropy (also known as Wiener entropy) is popular for quantifying harmonicity versus randomness and is widely used in birdsong analysis [25]. It can be understood as a measure of how distinct from flat the spectrum of an observed signal is. Harmonic stacks with high contrast yield low spectral entropy while random noise maximizes it. In speech, for example, voiced sounds produce formants while unvoiced speech tends to have a flatter spectrum. Background noise tends to reduce spectral contrast, but often formants are preserved. Spectral entropy is observed to help speech recognition in adverse conditions [69].

In ASR, spectral entropy has also been used to produce multi-resolution spectrograms, where adjacent frequency bands are combined to form a representation containing multiple levels of coarseness [70]. For a given spectrogram \( S(f, t) \), the spectral entropy for a band \([f_1, f_2]\) is

\[
H_{f_1, f_2}(t) = - \int_{f_1}^{f_2} df s(f, t) \log s(f, t),
\]

where \( s(f, t) = S(f, t)/ \int df S(f, t) \) serves as a spectral density distribution. To construct a representation at resolution \( J \), we partition the spectrogram’s bands into \( J \) non-overlapping subsets of equal sizes and apply any operation that integrates over each subset (e.g. the spectral entropy). For example, for 4 bands and \( J = 2 \), we have two
Figure 3.19. Multi-resolution spectral entropy. (A) The top panel shows spectrograms of four Bengalese finch syllables. The bottom shows their corresponding multi-resolution spectral entropies. Noisy syllables with broadbands are transformed into structures resembling harmonic stacks. True harmonic stacks become less defined. (B) Direct comparison of the true power spectrum and the spectrum of the spectral entropy for the two types of syllables. Both axes were normalized to facilitate the comparison. (C) Word error rates (WERs) for Bengalese finch syllable recognition when spectral entropy is absent and when it is added. This corresponds to tutor bird s933 in Fig. 3.21.

subsets of bands which result in a representation with 2 bands. A multi-resolution spectrogram is formed by concatenating multiple representations with varying $J$.

While many birdsong syllables are whistle-like and have clear harmonic structures, some are low-pitched and show a rougher quality with formant-like structures. The top row of Fig. 3.19A shows the spectrograms of four Bengalese finch syllables, where the first two have a distinct formant-like aspect. The row below shows the corresponding multi-resolution spectral entropies. Note that the formant-like structures are mapped onto stacks resembling harmonic stacks. Alternatively, real harmonic stacks tend to get scrambled. Fig. 3.19B compares the two representations in a normalized axis.

We have observed that feature detectors are often better at discriminating whistle-like syllables. It turns out that the addition of feature detectors trained directly on multi-resolution spectral entropy helps to discriminate low-pitched syllables. In Fig. 3.19C we compare the performance of two systems using the same sets of syllables. Entropy-tuned feature detectors added in tandem to regular feature detectors reduced the error rates in all training conditions.

3.3.6.2 Feature augmentation

In feature augmentation, we apply simple transformations to the training set in order to synthesize slightly different examples. It is reported to improve generalization in learning
algorithms, and it’s been particularly popular in neural networks. For example, in computer vision, it’s a common practice to flip, crop, colorize, change brightness, change contrast, jitter, and zoom in/out images. The transformed data is fed in tandem during learning. Some manipulations try to target specific perceptual effects. For example, partially cropping or masking objects in images is sometimes used to simulate occlusion. In audio, noise corruption, reverberation, masking, and temporal and frequency scaling perturbations are the most common transformations. Here we are particularly interested in techniques for frequency warping, pitch shifting and temporal warping. That is, transformations which reproduce variations in song production that could potentially be observed in real songs. In addition, we are also interested in transformations that help increase the robustness of feature detectors to background noises.

We found that small translations in the frequency axis can have dramatic effects in the responses of detectors. Note, however, that this is an unnatural transformation, since we do not account for the increase in the spacing between the harmonics. As illustrated in Fig. 3.20, adding copies of the training vectors after shifting up and down their frequency axes by at most 2 frequency bins makes the responses narrower. In some cases, however, it can suppress the responses to true positives (Fig. 3.20A, lower panel). This is accompanied by an increase in the number of true positives (Fig. 3.20A, lower panel). To understand how augmentation modifies the decision boundaries, we need to compare how the response of the system changes. In general, the STRF represent only the linear portion of the response of a system [71]. In the case of linear detectors, it is simply the weight vector \( w = \sum_i \alpha_i x_i^+ \) of Eq. 3.1. For the Gaussian kernel, we decided to directly sample the portion of the feature space contained within the decision boundary. Any point inside the decision boundary will drive the detector to respond and can be identified as a positive stimulus. For each decision boundary, we took its support vectors in pairs and calculated convex combinations for each pair. We then sampled the paths joining these pairs and kept only those which fully lied within the decision boundary. For example, if \( v_1 \) and \( v_2 \) are two support vectors, we sample \( t \in [0, 1] \) and collect the vectors \( v(t) \) of the form

\[
v(t) = tv_1 + (1-t)v_2.
\]  

(3.18)

The top row of Fig 3.20B shows samples of \( v(t) \) for the Song sparrow syllable to the left in the absence of frequency shifts. Note the similarity between each sample. The middle and the last rows show samples for the same syllable type following frequency shifts of 150 Hz and 615 Hz (support vectors not shown, only interior points connecting two
Figure 3.20. Feature augmentation with frequency translation. (A) Detector responses to two types of Song sparrow syllables before (left) and after (right) feature augmentation. Both positive and negative training sets were augmented with copies of feature vectors translated by up to 2 bins along the frequency axis. The responses after augmentation tend to peak higher (top panel) but can also become suppressed in some cases. (B) Feature vectors sampled within the decision boundaries suggest that augmentation modifies the decision region to include a progression of feature vector connecting the frequency-shifted copies.

support vectors, i.e. $t \in (0, 1)$). In summary, this type of feature augmentation forces the decision boundary to expand and include spectrotemporal patches that resemble superpositions of frequency-shifted versions of the positive vectors $x^+$. We speculate that this effectively causes the decision boundary to include broadband versions of $x^+$, leading to stronger responses to true positives. The suppression of true positive responses shown in panel 3.20A is likely due to a narrowing of the margins due to augmenting the negative set as well.

3.3.7 Continuous recognition of Bengalese finch syllables

Now we describe the performance of the spike-alignment method for Bengalese finch syllables in continuous recordings. In constructing and benchmarking our system, we selected a list of requirements to define success. First, we decided that the system needs to be practical for song annotation. It needs to be able to learn from a small set of annotations and generalize for unseen recordings of the same bird. This means that the generalization error needs to be low for small training set sizes. The error also needs to decrease when more training data is available. Additionally, the boundaries of the
predictions have to accurately identify the beginning and end of each syllable. Second, the performance needs to be evaluated based on the songbird species, not just on a single bird. For this reason, we show benchmarks with multiple Bengalese finches of different families. The same set of hyperparameters and routines should be used for all birds of the same species, so long the recordings can be assumed to have been made in similar circumstances.

A common challenge in measuring the performance of predictive models is deciding how to split the data into training, evaluation, and testing sets. Since we are interested in the performance using tens to hundreds of syllables for training, and we have access to thousands of syllables of each bird (see Table 3.1), it’s reasonable to measure the error just on a single randomly sampled test set. To address the concern of the choice of the training syllables, we run benchmarks on small training sets randomly selected from a larger pool of training syllables.

Fig. 3.21A summarizes the recognition errors for two families of Bengalese finches. Errors plateau at about 20 syllables per type for most individuals. As in the previous figures, we used introductory notes for training feature detectors but removed them before computing the WERs. We also completely ignored calls. Syllables which were partially cut at the end of the recordings were also removed. Although filler notes sometimes resemble introductory notes, we included fillers in both training and testing sets. Each point in Fig. 3.21A represents the average (bars show standard deviations) of 10 recognition systems trained on different sets. It’s important to note that we selected training and testing syllables from disjoint sets of recordings. A slightly less realistic point is that syllables in a training set may have been pooled from different recording days. This point may have helped the generalization performance across the entire set of testing recordings.

The WER measures the error of the output string but ignores the exact time and duration of the predictions. For the system to be useful, it needs to predict the correct onset and offset boundaries of syllables. Frame-based errors measure the prediction error at the frame level by comparing each predicted frame to manual annotations. The frame error rate (FER, the fraction of correctly predicted frames) accounts for time and duration, but it is not very insightful for continuous recognition since true positives from silence are typically the majority and saturate the frame accuracy rate. Instead, we calculated the F-1 score in the following way. If a frame annotated as silence was included as part of a prediction, we tagged it as a false positive (FP). Similarly, if it originally had a syllable label and the system missed it, we tagged it as a false negative (FN). True positives
Figure 3.21. Benchmarks with two families of Bengalese finches. (A) Word error rates (WERs) per number of training syllables for each individual bird. WERs are low enough in practice, even when only a few syllables per type are available for training. Only up to 20 syllables per type for training were available for benchmarking tutor bl5w5. (B) F-1 scores of frame error rates (FERs) for each bird show that the syllable boundaries predicted by our system are accurate and stable across all training conditions. (C) Predicted syllable boundaries deviate at most by tens of milliseconds from manual annotations. The panels in the bottom show examples of predictions for three renditions of the same song.

(TP) corresponded to frames which were correctly discovered and classified. Since we wanted to measure the accuracy of the boundaries of audio segments, we did not include misclassified frames containing vocalizations. The F-1 score is defined as the harmonic mean between the precision and the recall. It can be rewritten as $(1 + (FN + FP) / 2 TP)^{-1}$, which makes it explicitly that false negatives and false positives are put in the same footing as well as that it neglects true negatives (which in our case correspond to silent frames correctly marked as silence). Panel 3.1B summarizes the F-1 scores for the same two families of Bengalese finches. What is noteworthy here is that the quality of the boundaries is independent on the training set size. We also illustrate in panel 3.1C the distribution of boundary mismatch (the difference in the exact times of predicted and annotated syllable boundaries) and the typical output after recognition for Tutor bl5w5.
<table>
<thead>
<tr>
<th>Bird Id</th>
<th>Age (months)</th>
<th># types</th>
<th># syl.</th>
<th># test syl.</th>
</tr>
</thead>
<tbody>
<tr>
<td>s933 (tutor)</td>
<td>-</td>
<td>18</td>
<td>16,223</td>
<td>9,622</td>
</tr>
<tr>
<td>w27</td>
<td>15</td>
<td>13</td>
<td>25,977</td>
<td>16,060</td>
</tr>
<tr>
<td>w31</td>
<td>15</td>
<td>13</td>
<td>27,181</td>
<td>16,562</td>
</tr>
<tr>
<td>w34</td>
<td>4</td>
<td>10</td>
<td>35,206</td>
<td>21,190</td>
</tr>
<tr>
<td>w72</td>
<td>6</td>
<td>12</td>
<td>17,539</td>
<td>10,553</td>
</tr>
<tr>
<td>w74</td>
<td>6</td>
<td>9</td>
<td>22,485</td>
<td>13,604</td>
</tr>
<tr>
<td>w96</td>
<td>6</td>
<td>13</td>
<td>23,568</td>
<td>13,656</td>
</tr>
<tr>
<td>w100</td>
<td>6</td>
<td>19</td>
<td>14,950</td>
<td>9,230</td>
</tr>
<tr>
<td>bl5w5 (tutor)</td>
<td>-</td>
<td>11</td>
<td>6,318</td>
<td>3,990</td>
</tr>
<tr>
<td>br81</td>
<td>-</td>
<td>10</td>
<td>29,402</td>
<td>17,468</td>
</tr>
<tr>
<td>br82</td>
<td>-</td>
<td>12</td>
<td>51,290</td>
<td>30,985</td>
</tr>
</tbody>
</table>

Table 3.1. Summary of Bengalese finch songs used for the benchmarks in Fig. 3.21.

3.3.7.1 Limitations

Although the WERs and F1-scores in Fig. 3.21 are promising, our system has limitations. First, the assumption of fixed-duration prevents it from being used for syllables of species such as the canary and the song sparrow. Second, in our benchmarks, we have left out syllable types known to be more acoustically variable, such as introductory notes. Trills with short inter-syllable gaps also pose difficulties. Finally, it does not deal systematically well with cases where two syllable types share a large set of acoustic features, particularly when a syllable is composed of notes which are in turn also sung individually. In the following chapters we introduce modifications to deal with these issues.

3.4 Conclusions

In this chapter, we introduced a system for continuous recognition of birdsong syllables of fixed-duration. Our system works by converting the audio input into an ultra-sparse representation of feature-detecting units. Patterns embedded in this representation are discovered and reconstructed by a template matching algorithm. We benchmarked our system using the songs of 11 Bengalese finches. We found that the boundaries of the predicted syllables are accurate within roughly 50 ms, even the system is trained using 5 exemplars per syllable type. Boundary accuracy remains constant when the training set is increased. We also found that syllable labels are predicted accurately. The word error rate (WER) stays below 8% for 5 syllables per type in training for all birds. The WER decreases when more syllables are added in the training set. For some birds, the WERs
plateau below 1% at 10 syllables per type.

Two main limitations prevent us from directly using our system to recognize canary and song sparrow syllables. Namely, the fixed-duration assumption and the problem of note composition (that is, when two syllables share identical notes). In the next chapter we introduce modifications in the spike representation to deal with the fixed-duration assumption.
Chapter 4  
Continuous recognition of vocalizations of variable duration

In the previous chapter, we introduced a system for recognizing syllables of fixed duration. We made this assumption to avoid dealing with temporal stretching. It was also motivated by our practical constraints on the size of the training set. When we tried it on canary songs, we found that syllables containing harmonic stacks of variable duration were poorly recognized. Additionally, we also found that some syllable categories often shared acoustic features, making them difficult to distinguish. In Sec. 3.2.2.1 we addressed this issue by refining the negative training sets to reduce the load on the decision boundaries. In the canary, we found that this was insufficient and led to substitution errors.

In this chapter, we address these limitations. We start in Sec. 4.1 by reviewing some aspects of acoustic variability that make canary syllable recognition challenging. In Sec. 4.2 we introduce a spike-based representation for dealing with syllables of variable duration. Namely, syllables are now represented by sequences of onset and offset events. In Sec. 4.3 we review the spike-alignment graph introduced in the previous chapter and extend it to handle variable duration. We first attempt at canary phrase recognition, but quickly move on to syllable recognition, as we find it to be more accurate. In Sec. 4.3.3 we discuss handling acoustic variability and show how our modifications improve the recognition errors of Bengalese finch syllables from Ch. 3. When we try recognizing canary syllables, we find that the error rate is still large, mainly due substitution errors. To address this, we introduce the concept of binary detectors in Sec. 4.4. We show how Adaboost can be used to train binary detectors. In Sec. 4.4.4 we show how a system of feature detectors and binary detectors improves the recognition rate of both canary and Bengalese finch songs. We conclude by addressing the limitations of this system and set up the stage for song sparrows in the following chapter.
4.1 Acoustic variability in canary syllables

We saw in the previous chapter (Fig. 3.2) that canary syllables can have variable duration. In principle, if we had a sufficient number of annotated syllables, we could train enough templates to span all syllable durations. However, this is inefficient. Why did the system from Ch. 3 struggle with harmonic stacks of variable duration? Harmonic stacks carry only frequency modulation and no temporal modulation. Other syllables containing little temporal modulation and variable duration were also poorly recognized. In short, our approach of event detection fails for stationary sounds because these are not localized events. But, as we will see in the following section, their onset and offset times are.

When dealing with canary syllables, we also encountered difficulties from acoustic variations which we rarely saw in Bengalese finches. Some of the canary syllables we analyzed showed changes in pitch over the course of a phrase. We also found that some syllables sung in different days could vary in their modulation content. Fig. 4.1A shows two renditions of two syllable types annotated by experts and recorded over the course of two days [72]. Below the syllables we show their profiles of spectrotemporal modulation. Both syllables show a reduction temporal modulation from one song to the next. Both syllables also change in pitch. This change is such that the harmonics of one syllable type become more similar to the other type in a different song than itself (Fig. 4.1B). The statistics of simple acoustic measures for the two syllable types are also ineffective for discrimination. In Fig. 4.1C we show the spectral modulation content, median pitch and distribution of duration of the two syllable types for hundreds of songs. Note how they all overlap. This contrasts with the Bengalese finch syllables we saw in Ch. 3 and suggests that our changes also need to deal with more dramatic acoustic variations.

These are common problems in speech recognition. Pitch confers the speaker’s identity but doesn’t affect recognition. Vowels can stretch for arbitrarily long times and speech rhythm can change significantly without affecting comprehension. Temporal warping also happens in other perceptual modalities and is an important problem that organisms need to solve. Previous works have discussed possible solutions to temporal warping in networks of neurons. [73] proposes that warping invariance can be achieved by the spontaneous synchronization of the activity of spiking units in a network. [74] presents a bump attractor in a network of rate units which, collectively, achieve temporal invariance for spoken digit recognition. [75] describes a conductance-based model whose voltage potential warps according to the stimuli. The solution we describe here is target to extend the recognition system we developed in Ch. 3.
Figure 4.1. Example of acoustic variability in canary syllables. (A) Top: two syllable types in a canary repertoire sung in two different days. Bottom: magnitude of spectrotemporal modulation for each syllable type in each song. Red contour represent regions of 50% of the total power. The amount of temporal modulation in each syllable changes from one day to the next. (B) Changes in temporal modulation are also accompanied by changes in pitch. Syllable 12 in song 152 contains harmonics closer to syllable 13 in song 3628 than syllable 13 itself in song 152. (C) Over all songs, acoustic measures such as frequency modulation, pitch and duration overlap and, alone, cannot discriminate the two syllable types.

4.2 The on/off representation

The spike representation we introduced in Ch. 3 is useful when all the patterns within a category have similar duration. With the alignment-based method, patterns that warp uniformly can also be recognized. However, the representation becomes unsuitable when temporal warping is large or non-uniform. This can be conceptually understood from the following idealized scenario. Suppose we observe the sequence illustrated in Fig. 4.2A, generated by a rapid transient portion, a stationary part active only for brief period of time, and another rapid transient part. The transient portions are idealized consonant-vowel or vowel-consonant transitions, while the stationary part idealizes the vowel. Now we’re asked to evaluate whether the sequence in Fig. 4.2A represents an element within the same category. It is not obvious to tell without access to the subthreshold responses. This is because we do not have access to the total duration that each individual response...
Figure 4.2. Contrasting spike representations for time-varying syllables. (A) Representation used in Ch. 3, where we assign spikes to peaks in the detectors’ responses. The mid-portion represents a flat harmonic stack that warps temporally. The warped sequence resembles 3 isolated groups. (B) Onset and offset representation. Onsets and offsets come in pair and tile the entire duration of the syllable, even when it warps in time.

is representing directly from the sequences. For example, another interpretation of the pattern in Fig. 4.2A could be that it represents a consonant-vowel transition, followed by a brief silence, a short vowel, another brief silence, and finally a vowel-consonant transition. Of course, this example neglects the fact that a grammar would make one interpretation more likely than the other. But, for birdsongs, syntax of the song is not available to us until we have constructed it.

What this thought experiment suggests is that to represent patterns of varying duration with a small dictionary, we need at least three components for each dictionary element: whether the element is active or not, its start (or onset), and its duration. Equivalently, given the onset, the duration can be replaced by the end time (or offset). The first component can also be graded in order to represent intensity. Fig. 4.2B illustrates how this representation can help generalize to arbitrary durations. The initial portion of the onset sequence matches the observed one. However, the following portion is separated from the initial due the temporal warp in the middle. The offset sequence gives us the missing evidence to link the two onset portions.

4.2.1 Syllables as on/off sequences

Throughout Ch. 3 we used the SVM’s decision boundary as the threshold of detection. We also used the SVM’s lower margin as the lowest threshold allowed during sequence completion (Sec. 3.3.5.2). But the lower margin is close to negative vectors. In practice, we found that assigning onset and offset times to points where the responses crossed
the lower margin led to poor syllable boundaries, often merging adjacent syllables together. Placing on/off times at the crossing of the decision boundary, on the other hand, underestimated the syllable boundaries and frequently led to insertions. We resolved this dilemma by fixing another threshold between the lower margin and the decision boundary. We call it the on/off threshold (Fig. 4.3A). From the SVM decision function, Eq. 3.2, the decision boundary occurs for points where $y(t) = 0$. Similarly, the normalized functional margins occur at $y(t) = \pm 1$. Our on/off threshold is $y(t) = -1 + \delta$, where $\delta \in (0,1)$ is a constant. For our benchmarks, we fix $\delta = 0.2$. To summarize, we use the decision boundary for detection, which benefits from the margin robustness. We then trace the response starting from the detection point (i.e. the highest peak above the detection threshold) until it hits the on/off threshold, where we then place the onset and offset time points (Fig. 4.3B).

Canaries sing phrases consisting of trilled repeated syllables. Trills are also common in the song sparrow’s song. When the analysis window is much longer than the intersyllabic duration, it becomes hard to resolve individual syllables. Instead, the low resolution phrase may resemble a single continuous vocalization. This affects the on/off representation in two ways. First, if the small silent gaps between the syllables are visible and shorter than the detector’s integration window, the detector’s response will seem to oscillate above a certain plateau. Increasing the duration of the silent gap will eliminate this plateau. In either case, there is a chance oscillation will generate many on/off pairs, depending on whether it crosses the decision boundary. Also depending on the decision boundary, the sequence of on/off pairs may be inconsistent with the number of syllables. Second, suppose the detector is tuned for two elements within the trill, as opposed to one. Then, the first and last oscillation peaks will be shorter than the middle ones. To prevent these issues, we also force consecutive onsets and offsets to merge when their
gap is shorter than $\sim 13$ ms.

Onsets and offsets have a simple interpretation in terms of decision boundaries. The onset is the time when decision region is first visited and the offset is the time when it is exited. The warping invariance comes from stationarity of the input in respect to the interior of the decision boundary. As illustrated in Fig. 4.4A, temporal modulations, such as the downward harmonic sweep shown in the top panel, represent a brief visit to the decision region. The flat harmonic stack in the lower panel, on the other hand, spends a longer interval inside the decision region. Our syllable templates are now binarized sequences of onsets and offsets (Fig. 4.4B). In Fig. 4.4C we show examples of templates for Bengalese finch, zebra finch, and canary syllables.

In Fig. 4.5A-B we illustrate again how the on/off representation solves the stretching
Figure 4.5. Onsets/offsets and temporal warping in canary syllables. (A) A syllable consisting of an upward sweep followed by a harmonic stack of variable duration. The on/off representation stretches after the offset sequence warps at the beginning of the stack. (B) A syllable that stretches more uniformly. The on/off representation warps similarly.

This problem, this time with two canary syllables from different birds. In panel 4.5A, temporal warping occurs mostly for the flat harmonic stack following fixed-duration upward sweep. This causes the first few onsets and offsets to be stereotyped across all syllable exemplars. The subsequent onsets also show little variation amongst renditions. The offsets, however, right at the point of transition towards the flat harmonic stack become very variable, followed by stereotypy again. The temporal warping in panel 4.4B occurs more uniformly, which reflects the acoustics.
4.2.2 Reading the on/off representation

How do we read and bind onsets and offsets into sequences? Do we read all onsets first and then the offsets? Or do we read them concurrently? Or perhaps, as Fig. 4.5A suggests, we should read them in chunks according to their temporal variability? Should they be treated as independent sequences? In the next two sections we will explore two ideas for combining onsets and offsets.

In Sec. 4.3, we reuse the reconstruction method we developed in Ch. 3 chapter for recognizing onset sequences. We then recognize the offset sequence using the same method, but conditioned on the recognized onsets. Note, however, that our reconstruction method uses the sustained activity of feature detectors to perform search (In Ch. 3 we called it the search interval and assigned an auxiliary variable, $\xi$, to represent it. See Fig. 3.16). This sustained activity connects onsets to offsets. So, in reality, we’re implicitly using the fact that every onset is paired to an offset.

Although we observe low WERs for syllable recognition with small training sets, this system has limitations for neuromorphic applications. Implementing it with a network of spiking neurons would require units to sustain firing, which defeats the purpose of the on/off representation. We will address this limitation in Sec. 5.2 of Ch. 5, where we treat onsets and offsets independently. This means that we do not rely on the auxiliary variable $\xi$ for decoding. To allow for temporal warping, we assume that onsets and offsets cluster around temporal landmarks in syllables. Warping then is produced by the relative time between the synchronous activity of these clusters.

4.3 Search-based recognition

4.3.1 Spike-alignment graph revisited

The spike-alignment method we describe in Section 3.3.5.1 has limitations which become clearer under the scrutiny of more complex songs, such as the canary’s and the song sparrow’s. Because of our weighting scheme, the alignment graph may not have a unique longest path. This is not a problem in principle, but it may lead to suboptimal solutions when the input sequence has deletions. To understand why, consider the construction in Figs. 4.6A–C, where the second unit in the input responds only once while the template presumes that it should have responded twice. The graph construction in Figs. 4.6A and B successfully indicate possible candidates for the best path (we omitted some nodes for the ease of visualization), but the absence of weights for the last stage makes all
4 node candidates in Fig. 4.6C (and in total 4 paths) equally likely, whereas the two alignment nodes in the second column (that is, \(a_3^{(2)}\) and \(a_3^{(4)}\)) are evidently incorrect. We observe that, despite this ambiguity, it’s still common for our system to reconstruct most of the input’s spike sequence and make the correct final decision, provided there are no other competing sources, little intra-syllabic acoustic variations and little inter-syllabic common features (e.g. the Bengalese finch song). The major issue in selecting incorrect nodes in this case comes when we attempt sequence completion. In some few instances, there is a chance that alignment \(a_3^{(4)}\), for example, may incorrectly group onsets from neighboring syllables that share similar features. This can then lead to deletions and incorrect boundaries in the final prediction.

The second issue stems from the assumption that the first spike triggers the construction of the alignment graph. If the last unit responds first, the alignment will be poor (Fig. 4.6D). In practice, sequence completion may be able to reconstruct most of the sequence. In panel 4.6F we show an example of this issue for short portion of a canary phrase in the absence of sequence completion. Note the missing responses of the first two units for the initial 3 syllables. Finally, our grouping mechanism may incorrectly merge adjacent syllables when templates include long durations between onset and offset. In panel 4.6E, the final alignment merges an inserted response from the first unit with a perfectly matching pattern. Panel 4.6G shows this problem in practice for a sequence of song sparrow syllables. The template being matched here corresponds to the long trill. Before the trill start, however, the bird sings a very brief note resembling the acoustics of the individual elements in the trill. This in turn triggers template matching, which ends suffering from the effect illustrated in panel 4.6E.

To prevent these issues, we need to refine template matching. To help distinguish ambiguous alignment paths, we introduce a temporal warp penalty to the edges of the alignment graph. Keeping the notation introduced in the previous chapter, let \(a_i = (s_i, u_i)\) be an alignment node we selected to search for the onsets of unit \(k\). As before, we use the plateaus between the onsets and offsets of unit \(k\) (that is, \(\xi_i\) from Eq. 3.13) in the template to perform the search. Suppose a plateau with onset \(u_k = (d_k, t^{u}_k)\) finds the input onset \(s_k = (d_k, t^{s}_k)\), generating the node \(a_k = (s_k, u_k)\). The edge connecting \(a_i\) to \(a_k\) is then weighted by

\[
w_{ik} = w^0_{ik} \exp\{-|\Delta t_{ik}|/\tau_1\}. \tag{4.1}
\]

where \(w^0_{ik}\) is the original weight assigned to the edge, \(\Delta t = (t^{s}_k - t^{s}_i) - (t^{u}_k - t^{u}_i)\) and \(\tau_1\) is a penalty constant. Thus, the original weight is penalized by the relative time between the template’s onset and the input’s onset.
Figure 4.6. Ambiguities and issues in the alignment graph. (A-C) Example of the construction of a spike-alignment that leads to ambiguous solutions when there are deletions in the input. In A-B, the bottom path is more likely because it expands a larger number of nodes. But in C, the choice of the final node is ambiguous in the absence of weights, whereas nodes $a_3^{(2)}$ and $a_3^{(4)}$ are clearly poor choices. (D) The assumption that spike alignment is triggered at the first spike can also lead to poor solutions. Here an alignment graph skips spikes from two units. (E) Our light assumptions on temporal warping leads to incorrect grouping in the presence of insertions. Here the alignment graph incorrectly groups an insertion with the actual sequence. (F) Example of the issue with the first spike assumption for a canary trill. Here the first two spikes in the first syllable are missed. (G) Example of incorrect grouping in a song sparrow song. Here the very first note shown resembles the long trilled repetition in the end. Spike alignment merges all the syllables shown.

The fix the second issue, we need to force a backward search prior to the forward search. To ensure efficiency, backward search has a beam of 1 and collects alignment nodes greedily by the largest weight according to Eq. 4.1. Once the lowest unit’s response is found, forward search proceeds as usual.

Finally, to prevent incorrect merges due long-duration templates of flat harmonic syllables, such as the example of Fig. 4.6, we introduce a second warp penalty. This time it depends solely on the template. At each alignment node $a_i$, a memory trace sets a hard limit to how much the template can stretch until it spots the plateau $\xi_k$ of the next unit $k$. The condition we set for halting the search is that the time allocated for
searching the next onset or offset in the song, $t$, is

$$t - t^u_i < m(t^u_k - t^u_i).$$

(4.2)

Here $m$ is another global constant. For all of our results in the following sections, we fixed $\tau_1 = 17\text{ ms}$ and $m = 33\text{ ms}$.

### 4.3.2 Phrase recognition

In order to speed up recognition of canary syllables, we first attempted phrase recognition. As we’ve discussed, canary phrases consist of multiple, sometimes tens or more, syllables strung together by silent gaps. For most syllable types, this gap has low variance, and we can use it to help predict when the next syllable in a phrase occurs. But, for some syllables, the variance in gap duration can be large and increase with the variance in syllable duration (Fig. 4.7B). That is, the inter-syllable gap is less predictive for longer syllables. Thus, one question is how can we use the distributions of duration and gap duration for recognizing phrases. To approach this problem, we first assumed that the duration and the gap duration follow gamma distributions and estimated the parameters from maximum likelihood using Newton-Rhapson for optimization. It turns out that these fits are very similar to the normality assumption, which have a simple closed solution for maximum likelihood. Fig. 4.7A illustrates the fits for two canary phrases using different orders of magnitudes of songs. Note the small difference between the parametric models and the histogram for the first syllable type in regime of large number of observations.

The detection mechanism is the same as we outlined in Ch. 3. Recall that we use the average of the responses elicited by the delay grid in a low threshold regime, $\langle V(t)\rangle$, to detect syllable candidates (Eq. 3.16). The difference now is that we apply it to the onsets only. Next, we reconstruct the onset sequence of the first detected syllable. We then iteratively locate and jump from peak to peak in $\langle V(t)\rangle$ to find the following syllables. In Fig. 4.7C we show $\langle V(t)\rangle$ below the spectrogram of a canary phrase. There is a one-to-one correspondence between peaks in $\langle V(t)\rangle$ and syllables. Below, we show the spectrogram of $\langle V(t)\rangle$, which has an intense band around 18Hz, giving a total duration of approximately 56ms for syllable plus gap. Note in particular how this band is sustained even when pitch decreases significantly and the peaks in $\langle V(t)\rangle$ decrease in amplitude. Thus, we use the distribution of duration plus gap. For example, if the duration is $d$ and the gap duration is $g$, we estimate the next peak in $\langle V(t)\rangle$ using the distribution of $d + g$. 

75
Figure 4.7. Recognition of phrases in the canary. (A) Syllable duration (bottom) and inter-syllable duration (middle) distributions for two types of canary syllables using different orders of magnitudes of songs. Fits of two parametric distributions, normal and gaussian, to the histograms are shown. (B) For the exception to a few syllable types, syllables in canary phrases have correlated widths of duration and gap duration distributions. Timing between consecutive syllables in a phrase are easy to predict for the shorter ones. (C) The subthreshold activity of feature detectors show a steady frequency peak around 18 Hz, reflecting the rate of singing.

The normality assumption comes in handy here if we assume $d$ and $g$ are independent (although this is not the case in practice, it simplifies calculations). In this case, we simply use the interval $[\mu - \gamma \sigma, \mu + \gamma \sigma]$ to search for the next peak in $\langle V(t) \rangle$, where $\mu = E(d) + E(g)$, $\sigma = \sqrt{\text{Var}(d) + \text{Var}(g)}$ and $\gamma$ is the confidence level. We continue this process until we cannot find any new peaks. To determine the end of the phrase, we reconstruct the offset sequence of the last syllable. In our implementations of phrase recognition, we fixed the confidence level to 1 standard deviation with $\gamma = 1$.

Following phrase recognition we need to chop phrases into syllables and then, as in Sec. 3.3.5.3, delete potential insertions. Here we come into another dilemma, since we need to assign scores to syllables before we use the methods from Sec. 3.3.5.3. For phrases, it makes sense to use the ratio of elicited onsets to the number of discovered syllables. This ratio represents the average number of onsets per syllable. We then define
the phrase score as the fraction of the syllable onset ratio by the average number of onsets in templates:

\[
\frac{1}{N_s \langle |T| \rangle} \sum_{k=1}^{N_d} \sum_{t=t_1}^{t_2} 1(t \in ON_k)
\] (4.3)

where \(N_s\) is the number of syllables within the phrase, \(\langle |T| \rangle\) is the average number of onsets in the templates, \(ON_k\) is the set of onset times generated by unit \(k\) and \(1\) is the indicator function. However, it is not clear what score syllables should inherit from phrases. A simple solution is to pass on to syllables the score defined in Eq. 4.3.

To segment phrases into syllables, we use convex hull segmentation with the short-time power profile [76]. Convex hull segmentation requires us to fix a threshold for the maximal difference between the hull and the power profile. These are the portions where segmentation effectively occurs. Using the same threshold for all syllable types can lead to incorrect segmentation. Individual syllables in the canary, for example, may contain silent gaps longer than the inter-syllable duration of other syllable types. To estimate the hull thresholds, we collect samples of the maximum and minimum power of each syllable type. Then, we fix the threshold at the difference \(Q_{10}(E_{max}) - Q_{90}(E_{min})\). Here \(Q_{10}\) and \(Q_{90}\) are the first and last deciles, \(E_{max}\) is the maximum power and \(E_{min}\) is the minimum power.

### 4.3.3 Syllable recognition

#### 4.3.3.1 Dealing with acoustic variations

As we saw in the beginning of the chapter, canary syllables frequently show significant shifts in pitch in the course of the phrase. In Bengalese finches, syllables don’t usually change in much, but introductory notes can due to their noisy profile. In the method we outlined in Sec. 3.2.2.1 for clustering snippets of syllables, we force the system to select a reference syllable at the start. This reference syllable is compared with the other syllables in the same category to construct clusters of similar acoustic events. But consider the scenario where a syllable category contains two sub-categories, which we observe in particular with introductory notes in Bengalese finches. Depending on how different the sub-clusters are, our procedure may represent well one but not the other. The outcome is that the system may respond poorly to one sub-category. We propose two ways for patching this issue.

First, we try to ensure that our feature detectors tile their entire training sets. That is, at least one feature detector should be responsive at each time bin in all training
Figure 4.8. Feature detectors and acoustic variability. (A) Top: the plateaus $\xi_k$ created by feature detectors collectively tile the duration of the training syllable. Here $\xi$ represents the union of the plateaus. Bottom: when we increase the threshold for detection, the activity of the feature detector pool fails to cover the syllable. The red rectangle shows the gap. (B-C) Examples of Bengalese finch and canary syllables that are poorly represented by their own detector pools. To patch this issue, we train new detectors using the gaps flagged by $\xi$. The second rows of each panel show histograms of the number of training syllables containing gaps throughout the training process. Note how the histograms concentrate around zero at the end in both cases.

syllables. Fig. 4.8A illustrates this idea. At the top panel, three feature detectors respond consecutively, covering the entire syllable’s duration. Here we define the auxiliary variable $\xi(t)$ as the union of the individual plateaus $\xi_i$ from Eq. 3.13:

$$\xi(t) = 1 \left(t \in \bigcup_k \{t' | \xi_k(t') > 0\}\right), \quad (4.4)$$

where, as before, 1 is the indicator function. At the bottom panel in Fig. 4.8A, the last feature detector fails to respond, resulting in a gap in $\xi(t)$ over the course of the syllable. The gap is represented by the red rectangle. The gap’s presence then triggers the system to train new feature detectors using the syllable as a representative. The training process is the same as we outlined in Ch. 3.

We iterate this procedure by selecting a single syllable at each step. At the end of each iteration, we add the new detectors to the detector pool. This way, we avoid training an unnecessarily large number of units. Figs. 4.8B–C show this idea in practice for a Bengalese finch introductory note and a canary syllable, respectively. Due to acoustic
variability, the original set of detectors respond poorly for both syllables. The histograms to the right show the fraction of syllables with a remaining gap in the training set for at each iteration. Here the training sets for both syllable types include 30 syllables in total. Note how, in both cases, the histograms concentrate at zero at the end of the last iteration.

A second idea to deal with acoustic variability is to simply use multiple representative syllables to train a large set of redundant feature detectors. However, both ideas have the following drawback. With the increase in the number of detectors with potentially higher false negative rates, we need to lower the thresholds for the delay grids. The solution for this issue is pooling. That is, to group feature detectors in smaller pools that independently represent the target category. We will revise pooling in Sec. 5.2 of Ch. 5. For now, recall that the thresholds depend on the fraction of detectors of a syllable type. For all our results here, we fix the thresholds for all syllable types at 40% of the total number of feature detectors in each type.

4.3.3.2 The prediction stream revisited

So far we’ve used the average response $\langle V(t) \rangle$ of Eq. 3.16 and a fixed threshold $V_0$ for detection. For syllables that vary in duration, this average may flatten below $V_0$ and result in deletions. To address this issue, instead of averaging we will let templates discover their own detection candidates using their own delay lines. For each template $i$, we collect the largest peak above $V_0(t)$ in $V_k(t)$. We call them anchor points or anchors. Note that, previously, we’ve checked if the amplitude of $\langle V(t) \rangle$ crossed $V_0$ at each spike of the input. Here we are proposing to look for anchor points in $V_k(t)$ before the first spike is known. Therefore, to proceed with reconstruction, we first need to locate a spike in the sequence using the anchor point.

Fig. 4.9A shows a sequence of onsets elicited by a Bengalese finch syllable in black. On the same panel, the same sequence, but now aligned by a delay line in respect to the first spike, is shown in red. The synchronous response from this delay line, $V(t)$, is shown in the bottom in red. Note that it peaks around the first spike. To go from the peak in $V(t)$, the anchor point, back to the spike sequence, we define a window of duration $\Delta t$ around the anchor and expand it symmetrically along the time axis until a spike is found. In 4.9B, $\Delta t = 27\text{ms}$ ensures that 50% of the entire sequence is recovered.

No we need to estimate $\Delta t$ for each template. For that, first we pass all training syllable through the delay lines associated with each template. This is the same as illustrated in Fig. 4.9A-B. It returns a collection of responses of the form $V_k(t)$ for each
Figure 4.9. Anchor points for syllable segmentation. (A) Top: sequence of onsets generated from a Bengalese finch syllable (black) and the same sequence after it has been sent through a delay line. Bottom: synchronous activity, $V(t)$, associated with the aligned sequence of onsets. We call the peaks in $V(t)$ anchor points. (B) The flanks of the synchronous activity around the anchor point can be used to reconstruct the aligned sequence. Here a window of width $\Delta t = 27$ ms recovers 50% of the onsets. (C) Cumulative distributions of the fraction of recovered onsets with a symmetric window of width $\Delta t$ using the method outlined in panel B. Top row: onset and offset sequences of a Bengalese finch syllable. Bottom row: onset and offset sequences of a canary syllable.

4.3.3.3 Bengalese finch and canary syllable recognition

Before we continue with recognition of canary syllables, we re-evaluate the modifications we’ve made so far using the Bengalese finch song. In our evaluation in Sec. 3.3.7, we did not include introductory notes. In Bengalese finches, intro notes tend to be more challenging to categorize due to acoustic variability. But since we use intro notes for
Figure 4.10. Benchmarks for Bengalese finch syllables. Training conditions correspond to 5, 10, 15, 20, and 25 training syllables per type. For each condition we trained 10 recognition systems using syllables sampled independently and randomly.

training the detectors, it’s fairer to include them in the final evaluation as well. Fig. 4.10 shows WERs for the first family (s933 family) in Fig. 3.21A. For completeness, we also display the WER when intro notes are ignored from both annotations and prediction strings in blue. It turns out, the modifications we introduced help with generalization for intro notes. The difference in WER average when intro notes are included and when it is left out is small. The second family (bl5w5 family) in Fig. 3.21 had significantly lower WERs, so we decided to focus here only on the s933 family. The training conditions displayed in the horizontal axis correspond to 5, 10, 15, 20, and 25 annotated syllables per type for training and each condition shows the distribution of errors for 10 sets sampled independently and at random. To compare our results explicitly with existing recognition systems, we display the training conditions as the total duration of the set of annotated syllables used in training.

The WERs in Fig. 4.10 include errors from background noises in the recordings. Many of the recordings include noises such as wing flags, pecking, and steps. These sounds can sometimes introduce insertions, which can mislead us into believing that our errors are due only to misdetections in the song. To get a better sense of how our system performs in the absence of background noises, we also benchmark it for songs only. For each recording, we cut all the recorded silence before the first intro note (or first syllable) and after the last syllable. Fig. 4.11 shows this benchmark, carried in the exact same conditions as the previous one. Note how the WERs now approach zero. We also note a couple of jumps in some birds, for example one of the systems at 25 syllables in bird
Figure 4.11. Benchmarks for Bengalese finch syllables in the absence of noise. Here we benchmark only the portions of the recordings between the first intro note (or first syllable) and the last syllable. The conditions in this benchmark are the same as in Fig. 4.10.

Figure 4.12. Benchmark for canary syllables.

w34. Under inspection, we found that the training set for this particular system had been poorly chosen, which explains the low performance.

We then used the exact same set of configurations and hyperparameters to benchmark the canary we’ve been discussing so far. This canary had a significantly larger repertoire than the Bengalese finch syllables we’ve discussed. Some syllables were also significantly longer, making the training set longer. Fig. 4.12 shows the benchmark. Note how the error rate doesn’t decrease as we expected. It turns out the syllables in the canary song share many features and most of the errors come from substitutions. This suggests that our detectors are having trouble discriminating between syllable types. To address this difficulty, we introduce the concept of binary detectors in the next section.
4.4 Continuous recognition with binary detectors

In the previous section, we saw that syllable recognition in the canary song suffered from substitution errors. These errors come from feature detectors responding to multiple syllable types. To prevent them, we can force detectors to be more selective. But doing so may also force detectors to become unresponsive to unseen variations of their preferred syllable type (for example, variations as the ones shown in Fig. 4.1). In this section, we propose a solution for this dilemma. We do not force selectivity on feature detectors. Instead, whenever two groups of detectors respond vigorously at the same time, we relay the final decision to a group of specialized units. These units, which we call binary detectors, focus only on the important differences between the two syllable types while ignoring the rest. Together, feature detectors and binary detectors can segment and classify syllables accurately, even when they share common acoustic features. First we discuss the concept of binary detectors. Then, we show how Adaboost can be used for training. Finally, we benchmark Bengalese finch and canary songs and demonstrate that the addition of binary detectors resolves our problems with substitutions.

4.4.1 Binary feature detectors

Binary detectors address our concerns from the previous section. As the name suggests, these are units trained on two syllable types only and completely ignore other semantic labels. When two syllables share a large set of acoustic features, the role of binary detectors is to focus on specific features that discriminate the two syllables. In Fig. 4.13A, for instance, syllables A and B share the same onset, thus activating both pools of A and B feature detectors. The binary detectors shown to the right were trained to respond to the rightmost portion of A and remain silent for the entirety of B. Hence, we use the notation $A \setminus B$. As shown in Fig. 4.13A, they do indeed respond to the rightmost part of A and remain silent for B. Conditioned on the simultaneous activation of A and B, querying $A \setminus B$ effectively discriminates the two syllables.

Since binary detectors make decisions conditioned on feature detectors, they can completely ignore large chunks of the feature space and focus entirely on the relevant parts for binary classification. This property is helpful because it allows us to train binary detectors using very simple methods, which in turn helps with generalization when data is scarce. For our purposes of classification, we can picture this idea as in Fig. 4.13B: ignoring most points in the feature space allows for a much simpler decision boundary.

In Fig. 4.13C we show an example of binary classification during continuous recog-
Figure 4.13. Binary feature detectors. (A) Binary detectors easily solve the problem of two competing syllable types by responding only to the differences between the two syllables. (B) Binary detectors by design ignore most of the feature space and focus on features that are essential for discriminating between two categories. This leads to simpler decision boundaries. (C) The responses of binary detectors are conditioned on the simultaneous detection of the two categories that it discriminates. Therefore, binary detectors may respond to other syllables without affecting recognition.

The highlighted part of the spectrogram is the union of two predictions (the two Bengalese finch syllables shown previously in Fig. 3.8). The binary detectors are active for the second half of the syllable and correctly classify it. Note that they are also activated by the neighboring syllables, but are only queried and participate in the decision for the highlighted one.

The examples in Fig. 4.13A and C suggest that binary detectors can help when we know that two syllables share a set of features. However, in practice, we first need to discover these common features. For the note composition problem, that is when syllables share the same note, we can use feature detectors to discover the common notes. We will explore this problem in Ch. 5 using the song sparrow’s song. For the canary song, we use a more brute force approach to train ensembles of specialized binary detectors. In the following sections we’ll see how Adaboost helps us select exemplars from which we extract patches of spectrogram and train binary detectors.

4.4.2 Boosted binary detectors

Adaboost is an iterative meta-learning algorithm that constructs a classifier based on an ensemble of weak base learners [77]. The fundamental question it addresses is: in what circumstances will an ensemble of weak learners work as a strong learner? To solve this problem, Adaboost iteratively builds a distribution of weights $D(x)$ over the set of training examples $x$ from the misclassification errors of each learner. Examples that are
harder to classify receive larger weights. This way, Adaboost is able to discover and focus on the important examples that make classification difficult. The training error is guaranteed to decrease exponentially within finitely many iterations [77]. Here we are interested in binary classification with a threshold function.

The learner we have been using so far is not weak in the formal sense. Although we could still use it in principle, boosting a strong learner may be counterproductive. But, most importantly, SVMs can be slow to train. The main reason why Adaboost is attractive to us is that it can construct $D(x)$ at the same time it learns a strong classifier from weak ones that are simple to train. This involves training and sampling errors of multiple learners in each iteration. Our plan in this section is to train binary detectors for pairs of syllable types, which can be a large number in practice. Thus, we will restrict our discussion to training via gradient descent.

As we saw in Ch. 2, minimization of the Hinge loss function with L2 regularization by gradient descent approximates the large margin solution of SVMs. We use this strategy in this section. We let Adaboost discover the exemplars of syllables which are the hardest to classify. That is, we construct the distribution $D(x)$ over the set of training syllables (here $x$ refers to a syllable). Our SVM learns to discriminate patches in the spectrogram of $x$. This is similar to the face recognition algorithm introduced by [78].

### 4.4.3 Binary ensembles of spectrotemporal patches

We consider binary classifiers of the form

$$f(x) = \Theta \left( \sum \alpha_i h_i(x) + b \right),$$

where $\alpha_i$ are weights, $h_i$ are linear decision functions and $\Theta$ is the Heaviside step function. Given a fixed number of decision functions $h_i$, our task is to learn the hyperplanes defined by $h_i$ and weights $\alpha_i$ such that two syllable categories can be well discriminated by $f$.

First we define what $x$ in Eq. 4.5 represents. Suppose we have two classes of syllables, $A$ and $B$, each containing $N$ syllables. Each syllable $i$ corresponds to a $F \times T_i$-dimensional spectrogram. How do we visually distinguish $A$ from $B$? There are three cases: syllable $A$ may not resemble $B$ at all, $A$ may resemble $B$ in some aspects, or $A$ may contain an entire chunk of $B$ (or vice versa). We can assume that the first scenario has been taken care of by the first layer of feature detectors, so we only need to look at the second and third cases. We will explore the third case in Ch. 5. In this section we focus on the second case, which is common in the song of the canary.

85
We address this problem by letting binary detectors focus on specific spectrotemporal patches, instead of forcing them to integrate over a fixed window size and across the entire frequency axis. This way, binary detectors can later act as specialized attention units. So let $\Delta t_{\text{min}}$ and $\Delta t_{\text{max}}$ be the minimum and maximum durations allowed for a patch. Similarly, let $f_{\text{min}}$ and $f_{\text{max}}$ be the boundaries in frequency. In practice, we assume that $\Delta t_{\text{max}} < \min_i T_i$. We can also think of the patch as a mask of 1’s within the patch and 0’s everywhere else in the duration of $\Delta t_{\text{max}}$. This makes it easier to implement binary detectors reusing the framework of our feature detectors. Thus, we can write

$$h_i(t) = \omega_i \cdot x(t) + b_i,$$

(4.6)

where $x(t)$ is a vectorized window of the spectrogram with duration $\Delta t_{\text{max}}$. The weights are $\omega_i = w_i M_i$, where $M_i$ is a sampled mask and $w_i$ are the tunable weights.

Now, if we have $N$ training syllables of the same duration $T$, there are exactly

$$n_i = N \sum_{t=\Delta t_{\text{min}}}^{\Delta t_{\text{max}}} \sum_{f=f_{\text{min}}}^{f_{\text{max}}} (T - t + 1)(F - f + 1)$$

(4.7)

patches in total that we can query from the training set. This can be an absurdly large number in practice. Which patch should we pick? The answer is not always clear by visual inspection, so we need to go by trial and error. This is where Adaboost is most helpful.

We now summarize the algorithm. Let $x_i$ represent syllables in the training pool and $y_i \in \{1, 0\}$ be their labels (so $y_i = 1$ for the positive syllable and $y_i = 0$ for the negative). We initialize normalized weights as $D(x_i) = 1/N$, where $N$ is the size of the training pool. From the training pool, we select a few exemplars of both syllable types at random. Next, for each positive syllable $i$, we sample at least one spectrotemporal patch at random. This is easily done by first sampling a point within the interval $[T_i, F]$ (recall that $T_i$ is the duration of syllable $i$) and then the duration and frequency band such that the patch lies within $[T_i, F]$. Then, we train each sampled patch against one negative syllable and measure the error in respect to the training set. These steps are illustrated in Fig. 4.14A-C. Note from the histogram in 4.14C that many of the trained patches may achieve the same error rate. For the next step it’s sufficient to select one detector with the smallest weighted error and add it to the ensemble (Fig. 4.14C and D). This last step
Figure 4.14. Boosted binary detectors. (A) Exemplars of two Bengalese finch syllable types used for training. (B) At each step, a subset of syllables is sampled according to a distribution $D$. At the first step, $D$ is initialized as the uniform distribution. (C) Spectrotemporal patches are sampled at random from the syllables in panel B and used for training linear binary detectors. Their performance is evaluated on the entire training set. (D) In each iteration, the best performing detectors is added to the ensemble. (E) At each iteration, The distribution $D$ is updated according to the ensemble’s performance on the training set. The update follows the Adaboost rule. Syllables that are harder to classify get boosted. (F) Summary of the training process. (G) Ensemble of binary detectors respond to the positive syllable $A$ and remain silent to $B$. (H) Training and test error at each iteration. Adaboost prevents overfitting even when training error is zero.

solves the following query problem of [78]:

$$
\epsilon_t = \arg \min_h \sum_i D_t(x_i) \mathbf{1}(h(x_i) \neq y_i).
$$

Finally, we update the weights and according to

$$
D_{t+1}(x_i) = D_t(x_i) \frac{\exp\{-\gamma_t h(x_i)y_i\}}{Z_t},
$$

where $Z_t$ normalizes the expression and turns $D_{t+1}$ into a distribution. Here the update
equation 4.9 and $\gamma_i = 1/2 \log(1/\epsilon_i - 1)$ can be directly derived from the Bayes optimal decision rule with the assumption that the detectors’ predictions $h_t(x_i)$ are independent of each other when the true label $y_i$ is given [77]. In practice, we use $\gamma_i = 1/2 \log((1 - \epsilon + e_0)/(1 + e_0))$, with $e_0$ a small constant, to prevent zero division error. Fig 4.9E shows the updated $D(x_i)$ over several iterations. In the course of training, the weights redistribute and favor sampling syllables that result in higher error rates. After the weights are updated, the algorithm iterates over. Fig. 4.9F summarizes each iteration. Trained binary detectors respond to one syllable type and remain silent to the other (Fig. 4.9G). Another helpful feature of Adaboost, as illustrated in Fig. 4.9, is that the test error does not increase when the training error plateaus and in some cases even decreases. This has been explained by the margin bound [79].

Fig. 4.15 illustrates training and test performances of binary ensembles to two canary and one Bengalese finch syllable. Training sets had 30 syllables of each type. Test errors were computed to about 200 exemplars of each syllable. The canary syllables in Fig. 4.15A are similar and commonly mistaken by feature detectors. Adaboost reduces the test error to under 4% in less than 24 iterations. Figs. 4.15B and 4.15C show the evolution of the weight distribution $D$ and error distribution of the ensemble during training. For the canary syllables in Fig. 4.15D, test error drops below 3%. The Bengalese finch syllables in Fig. 4.15E are broadband, noisy, and commonly misclassified by feature detectors.

### 4.4.4 Syllable recognition with boosted binary detectors

We first reevaluated our benchmarks from Sec. 4.3.3.3 using boosted binary detectors. In Fig. 4.16 we show the WERs for the same Bengalese finch families we discussed before. In Fig. 4.17 we show the error rate for the same canary we benchmarked in Fig. 4.12.

We next benchmarked our system using Bengalese finch syllables recorded by a different laboratory [18]. The songs of this lineage have significantly different properties. On the level of the syntax, these birds frequently repeat syllables successively, resembling short phrases. Their vocabularies are also smaller. On the acoustic level, we observe that their syllables, and particularly the repeated syllables, are strung together by shorter silent gaps. In some instances, their introductory and filler notes seem to allow for some amount of temporal warping. These differences are useful because they allow us to assess our system’s performance in slightly different conditions. This is important because it helps us to pinpoint where our system fails to generalize in practice. For example, in our results in Fig. 4.16, we see that bird w27 generalizes extremely well for the hyperparameters we chose. Bird w96, on the other hand, shows larger WER fluctuations.
Figure 4.15. Training and empirical errors of boosted binary ensembles. Training sets contained 30 syllables of each type. Test set contained 200 syllables of each type. (A) Two canary syllable types commonly misclassified by feature detectors. (B) Evolution of the Adaboost weights given to syllables during training. (C) Distribution of classification error of the ensemble during training. (D) Two other commonly misclassified canary syllables. (E) Two broadband and noisy Bengalese finch syllables.

Figure 4.16. Benchmarks for Bengalese finches using binary detectors
This raises whether our system can recognize syllables of other lineages or if we simply found a fine-tuned solution for one specific family. Fig. 4.18 shows WERs for 8 out of the 11 birds in the dataset provided by [18]. For 3 of the 11 birds, we found the manual annotations provided with the data did not match the criteria for training our system, and we decided to not include their benchmarks here. We also highlight in blue the contributions of insertions to the total WER for each bird.

### 4.5 Conclusions

In this chapter, we described modifications in our automatic syllable recognition system. These modifications allow our system to accurately recognize the entire repertoire of Bengalese finch and canary syllables in continuous recordings. We introduced an ultra-
sparse onset/offset representation to deal with the problem of temporal warping. We also introduced binary detectors to help distinguish syllable types when they share acoustic features. We showed that our system generalizes well under very limited training data and that the generalization error decreases when more syllables is added for training. For Bengalese finches, the WER averages about 6% when 5 syllables per type are used in training. For the canary, the WER starts at about 10% at 5 exemplars and plateaus at about 6% at 20 exemplars.

How do these benchmarks compare to existing technology? In the last decade, modern deep neural networks have been proven versatile tools for classification using large datasets. A recent work using DNNs for birdsong syllable recognition reports WERs averaging at about 4% for Bengalese finches when 30 s of training data is used in training [20]. At 300 s, they average to about 0.9%. For the canary, their WER starts at about 11% using 190 s of training data and approaches 5.2% when the training set increases to 660 s. Note that the duration of our training set for the canary is about one order of magnitude shorter. This trade-off in accuracy and training set size suggests that the two systems are complementary and can be useful in different occasions. The system described in [20] has recently been used to help in identifying neurons in the avian brain responsible for keeping a memory of syntax transitions over long time scales [72]. Cohen and colleagues trained their system on 100 manually annotated songs and used it to automatically recognize a large amount of recordings. Our system, on the other hand, can be extremely helpful in the early stages of annotation, when only a few songs have been manually curated.

But, most importantly, our system represents a novel approach to continuous recognition of vocal elements. Our ultra-sparse representation and binary spikes are generated by independent units in parallel. Template matching is also carried in parallel. By relaying template matching to multiple processors concurrently, we can drastically decrease running time. In particular, we believe that it may represent an important application in neuromorphic computing.

Does our system generalize to more complex songs? The song sparrow represents another step up. The main difficulties introduced in song sparrow songs are the large repertoire (more than 50 syllable types) and note composition. We will address these problems in following chapter.
Chapter 5
Continuous recognition and compositionality in birdsongs

In the previous chapter, we introduced the concept of binary detectors to help in recognition when syllable types share acoustic features. What we did not discuss in detail, however, was the case when two or more syllables share entire notes. As we will see in this chapter, note composition occurs in Bengalese finch songs and, more frequently, in the song of the song sparrow.

In speech, the ubiquitous presence of coarticulation in phonetic sounds has led researchers to adopt contextual units such as diphones and triphones. Birdsong syllables don’t show the level of coarticulation found in speech, but notes can be shared amongst syllables. In most cases, these composite syllables resemble a concatenation of two or more notes. Sometimes a small silent gap between the notes may also occur. There are also cases when syllables don’t share notes, but share large sets of identical features. One example is a target pitch sustained for a long time in two syllable types (see Fig. 4.7 for an illustration in the canary). For the task of automatic recognition, we need to automatically identify the note repertoire or the set of shared features.

In Ch. 3, we circumvented this problem by adopting a few heuristic classification rules. These included directly comparing duration, power spectrum, and the maximum of cross correlation. This approach can be brittle in practice. For example, syllable duration can be variable in the canary song; power spectrum integrates over time and characterizes poorly frequency sweeps; the maximum cross-correlation may be small when pitch changes even slightly. When classes share many features, the method outlined in Sec. 3.3.5.3 becomes unreliable. In Ch. 4, our binary detectors solved the related problem of feature composition. We used an ensemble of binary detectors to distinguish between pairs of similar syllable types. In this chapter we develop a few different methods
to tackle the composition problem directly.

In the first part, we discuss using binary detectors to resolve syllables into subunits. Recall that feature detectors first segment the song and send a stream of predictions to binary detectors. Binary detectors rely on the segmentation and only need to solve the problem of isolated recognition. Here we develop a method for discovering shared patterns amongst syllable types. We use these patterns to train binary units and show how it helps with generalization in the song sparrow song.

In the section that follows, we develop a different strategy to solve simultaneously the temporal warping problem and the note composition problem. We first revisit continuous recognition using the concept of pooling. To construct pools of feature detectors, we cluster them based on their response patterns. Finally, we explore how pooling can help in syllable segmentation.

5.1 Recognition in the presence of shared subunits

To start, let us consider the simplest version of the problem. Suppose we have two syllables, $A$ and $AB$, with $A$ and $B$ denoting individual notes. As in Ch. 3, we train feature detectors for both syllables types, giving us a number of $D(A)$ and $D(AB)$ detectors for types $A$ and $AB$, respectively. During recognition, a syllable of type $AB$ will simultaneously activate a fraction of $D(A)$, $N_A = n_A D(A)$, and another fraction of $D(AB)$, $N_{AB} = n_{AB} D(AB)$. In turn, the activation of these units will produce predictions for $A$ and $AB$ with scores $n_A$ and $n_{AB}$, respectively. Under the assumption that no units are specialized for the coarticulation between notes $A$ and $B$ within $AB$, the scores $n_A$ and $n_{AB}$ will be close. When insertions and deletions are factored in, these scores become unreliable. Therefore, we need extra steps to decide which prediction to keep and which to discard. Note that there is nothing particularly important about the order of the notes in $AB$, as we could make a similar point regarding $BA$. In fact, a general solution should also account for the possibility of both $AB$ and $BA$.

5.1.1 Rule-based decisions

Ideally, we would have access to some sort of phonetic transcription of the song in terms of notes. This would prevent us from discriminating between two identical notes and help reduce overfitting. For example, when training for $AB$, we wouldn’t force detectors to discriminate against $B$. For some birds, this is a feasible but time-consuming manual
task. In our early attempts, we used a cascade of simple rules to select the correct predictions from the prediction stream. This proved to reduce the WERs substantially. For example, if $ABC$, $AB$, and $A$ are all detected, then we know that $A$ exists. Next, we look for the responses of detectors in $AB \setminus A$. We use the notation $X \setminus Y$ to identify the portion of $X$ that doesn’t include $Y$. If $AB \setminus A$ has no spikes, we confirm the identity of $A$. Else, it suggests that the syllable may either be $AB$ or $ABC$. A final step allows us to find the syllable type. For more elaborate cases we could construct finite state machines to accept strings based on the presence or absence of specific detectors in the sequence. For the birdsongs we study here, this is not necessary.

As far as these examples are concerned, compositionality is only an issue at the feature level. Different from the $AB$ versus $B$ case, feature compositionality may not be obvious, so we do not discuss the possibility of manually annotating it. In fact, it may also be subjective for feature detectors if they lack the ability to resolve at the necessary level of detail. As an example, consider a fast trilled repetition (e.g. the song sparrow trill in Fig. 4.6G) versus a harmonic stack with similar harmonic content as the trill. The two syllables can only be discriminated by a detector when it has access to features at a resolution larger than the duration of the individual elements in the trill. We discussed in Sec. 3.2.2.1 how to prevent complex decision boundaries when syllables resemble each other. Our binary detectors of Ch. 4 dealt more methodically with the problem when a few features are shared.

In what follows, we discuss training note composition rules when no transcription is available. We then discuss training binary detectors to implement these rules. In the last part, we briefly reprise boosted binary detectors.

5.1.2 Transition detectors

One feature that distinguishes $AB$ from $BA$ and also from the individual notes $A$ and $B$ is the context at the endpoints of each note. The individual notes $A$ and $B$ are surrounded by silence. $AB$ and $BA$ are also surrounded by silence, but between the individual notes there are also the transitions $A \rightarrow B$ and $B \rightarrow A$, respectively. The detection of one of these transitions is sufficient to distinguish the syllable type. However, in the general case, where strings such as $ABA$, $BAB$, etc. also exist, we need to detect the initial and final transitions from silence to the initial note and from the last note to silence.

To proceed, we need to define what a transition is in the spectrogram and train units to detect it. In birdsongs, notes within syllables are most commonly identified either by
silent gaps or fast spectral changes. Silent gaps are easy to estimate from the valleys in the loudness profile using a threshold. We can simply select the midpoint between the two points of threshold crossing as the transition. The transition detector’s span of integration has to be longer than the gap’s duration.

In practice, this idea has several drawbacks. First, a fix threshold for one syllable type may not work for another type. This may mislead the program into breaking syllables that do in fact contain short silence gaps into notes, particularly when the dataset is unbalanced and only few examples of that syllable type are available. Second, it disregards notes connected by fast spectral changes. Third, we may need to adapt the duration of integration of the transition detector according to gap size for each syllable type. Not doing so may force the system to train units to detect silence only. Last, it ignores the extra context that exists surrounding the transition point itself.

A more robust idea is to define transitions using not the spectrogram or the waveform, but the responses of the layer of feature detectors. The idea that the representation generated by one layer is reused to teach the next layer has been notably popularized by deep belief networks, where a stack of hidden layers is learned sequentially [80]. Our approach differs in that there is no representation learning. We use the representation created by the first layer, the prediction stream, to teach units in the second layer to attend to specific patterns of the input. In the strict sense of representational learning, our system is not multi-layered. We will nevertheless still use the term layer as a reference to the decision flow in our system.

Consider, for example, the output of the prediction stream for the snippet of song sparrow song in Fig. 5.1A. The solid vertical lines mark the prediction for syllable $AB$ and the dashed lines for syllable $A$. Syllable $A$ is a part of $AB$ and also appears in other contexts for this bird. Here the first layer identifies both syllable $AB$ and the note $A$ within $AB$. The first two columns in Fig. 5.1B illustrate heuristic rules to identify the transitions when both $A$ and $B$ notes exist on their own. We first identify the order of the overlap between the two predictions. The overlap condition is given by Eq. 3.15 with $\alpha = 0.5$. In the first row, for example, $B$ overlaps with the end portion of $AB$, so the onset of $B$ represents the transition. Similarly, $A$ overlaps with the beginning of $AB$, so the transition occurs at the offset of $A$. The third column illustrates the case when the mid-portion is also an individual note, $C$. In this case, we have two transition regions which are treated independently.

This strategy for detecting transitions between notes still falls short in that it ignores the additional context surrounding the transition point. The reason why this is relevant is
that the gap duration can be variable by small amounts. A single unit may not generalize well in practice. To extend it, we adopt the concept in Fig. 5.1C, in which multiple feature windows can contribute to distinguish between AB, A and B. For example, in the first column of Fig. 5.1C, the entire interval where AB and B don’t overlap can be used to train units to distinguish between AB and B. But while the feature windows that compose note transitions are unlikely to be found in other syllable types, other feature windows in the surroundings may not be so unique. We’ll address this issue in the next section by introducing binary detectors.

In using the first layer of trained detectors to discover notes in the repertoire, we need to make clear the distinction between training and held data. Feeding back the data used for training the first layer to itself will produce a robust prediction stream.
with high scores. Syllables sharing notes will robustly activate all detectors trained for those notes. This reflects how we selected the acoustic events to train the detectors (See Sec. 3.2.2.1). Doing the same with held out data, on the other hand, may produce additional predictions for syllable types which only resemble each other. As an example, compare panels A and B in Fig. 5.1. Panel A was obtained with held out data. Syllable E, which precedes syllable AB, despite only a vaguely resembling AB, is itself activated by AB. Panel D shows a series of training syllables artificially concatenated between silent intervals (i.e. not an actual song produced by the song sparrow). Here, G, H, and GH are all individual syllables in the bird’s repertoire. The prediction scores are all high. Additionally, although syllable J contains features that resemble G, it does not activate the detectors trained for G.

5.1.3 Binary detectors

Here we revisit the idea of binary detectors introduced in Ch. 4 with the goal of training decision rules. As a rule of thumb, we train binary detectors to respond to feature windows in the longest syllable. This takes care of the case when AB exists in addition to either A or B. When all three syllables exist, as in Fig. 5.1D, two sets of binary detectors, $A \setminus B$ and $B \setminus A$, need to be trained. Since decisions in the prediction stream are carried two-by-two (see Sec. 3.3.5.3), $A \setminus B$ and $B \setminus A$ are queried sequentially in arbitrary order. For example, $A \setminus B$ first selects $AB$ over $B$ and then $B \setminus A$ selects $AB$ over $B$.

With feature detectors and binary detectors, we can also train simple decision rules for identifying syllables that intersect in the feature space. As we outlined in Sec. 5.1.2 and illustrated in Fig. 5.1D, first we feed feature detectors with their own training syllables. Next, we let the prediction stream identify, according to Eq. 3.15, all the competing predictions. We illustrate with an example using a Bengalese finch in Fig. 5.2A. The first note, A, doesn’t exist by itself in this bird’s repertoire, but we segmented it from AB for the sake of illustration. The red rectangles indicate the union of the temporal span of all predictions. Their large overlap and high scores flag the training of binary detectors.

Take A versus AB first. AB is longer, so it has precedence. It immediately flags the training of binary detectors to discriminate $AB \setminus A$ against A (see Fig. 5.2B). We call detectors in this category $AB \setminus A \times A$. Note that the alternative, $A \times AB \setminus A$, would also work here. But it should be clear why we selected the longer syllable first: as $A \times AB$ would imply discriminating A versus itself, which would be a poor choice. A also competes with AC, so binary detectors $AC \setminus A \times A$ are also trained. Finally, to
Figure 5.2. Decision rules with binary detectors. (A) Feature detectors respond robustly to their own training set. Their responses generate competing predictions, which tell us which set of decision rules are required for discrimination. (B) For the predictions in panel A, only 3 sets of binary detectors are necessary for discrimination. In our notation, $AB \setminus A \times A$ represents binary units that respond to $AB \setminus A$ and remain silent for $A$. (C) A simpler set of rules can solve feature compositionality. Here we use held out data to identify feature compositionality. The two syllables share the start. We train binary units $E \times D$ or $D \times E$ to discriminate them.

decide between $AB$ and $AC$, the system may train either $AC \setminus A \times AB$ or $AB \setminus A \times AC$, depending on whichever is the longest.

This discussion covers the training of hard set rules for note compositionality. For feature compositionality, we use held out syllables. Here rule training is simpler because of our assumption that there are only a small set of shared features. We simply train binary detectors to discriminate one syllable versus the other. In Fig. 5.2C, for example, syllables $D$ and $E$ share a start. This prompts the system to train either binary units of the form $E \times D$ or $D \times E$, depending on their durations.

One potential concern is when there are many syllable types. The song sparrow we’ve used in our discussion, for example, has a rich repertoire with more than 50 syllable types, many containing concatenated notes. In the case when evaluation data isn’t available, it’s tempting to attempt to train binary detectors for every pair of syllables. For the song sparrow, this brute force approach would train thousands of binary detectors. In practice, we let the system train binary units only when prompted by the predictions stream.

### 5.1.4 SVM-based binary detectors

We discussed the notion of binary detectors and how they can help during recognition. Here we discuss some details in our implementation of a rule-based system of SVM binary detectors for syllable recognition. In our benchmarks, we trained binary detectors with hard-margin linear SVMs. Here we had to reduce the size of the training set in order to save data for a held out set. We fixed $4/5$ of the data for training feature detectors and
held the remaining $1/5$ for later training binary units for feature compositionality. To train binary units for syllable compositionality, first we fed feature detectors with their own training syllables and collected predictions pairs that competed in the prediction stream according to Eq. 3.15. We then filtered out predictions with scores less than 0.25 to retain only instances representing syllable compositionality. From these predictions, we selected positive sets of training features according to our discussion in Sec. 5.1.2 and illustrated in Fig. 5.1C. We used each of these positive feature vectors to train one binary detector. As in Sec. 3.2.2.1, we also supplemented the positive sets of binary detectors with feature vectors collected after clustering with the cross-correlation method (Fig. 3.7). Negative sets were also selected as outlined in Sec. 3.2.2.1, but with a higher value for $\cos(\Omega_{\text{min}}) = 0.99$. No silence of noise instance was included.

In a second stage, we also trained binary detectors for feature compositionality similarly. This step included only pairs of syllables which were not used for training binary units in the first stage. The main difference was that we fed held out data, as opposed to training data, to feature detectors to obtain the competing predictions. We also lowered the required scores to 0.1. Fig. 5.3 illustrates the steps we take for decoding a song sparrow syllables with decision rules and binary detectors in practice. The syllables shown in Fig. 5.3A are all part of the bird’s repertoire. Curiously, the portion connecting $A$ to $B$ in $AB$ resembles coarticulatory effects in speech. Fig. 5.3B shows the onsets of feature detectors trained to each type. The sequences are all complete and score high. The offsets are not shown, but the predictions all intersect in the prediction stream. Here we use a score margin of 0.25. The score differences between each prediction pairs are less than this margin. This in turn calls two pools of binary detectors, $AB \setminus A \times A$ and $AB \setminus B \times B$ (Fig. 5.3C). Both pools respond vigorously (only onsets shown). First $AB$ is compared and wins against $B$, then $AB$ is compared and wins against $A$. We set a minimum score of detection of 0.5 for binary pools. In 5.3D we show the prediction stream before and after the comparisons.

In practice, we found that binary detectors trained in this way reduced the number of substitutions, which was our original goal. But we also observed a slight increase in the number of insertions and in some cases deletions as well. For instance, in one early benchmarks with song sparrow syllables, we used 5 syllables per syllable type for training (for the songs we analyzed in this particular bird, we found 56 different syllable types, so this corresponds to a total of 280 syllables for training), and the introduction of binary detectors reduced the WER from 16.6% to 11.6%. This benchmark included 1275 test syllables. However, at same time, the insertion error rate went from 1.7%
Figure 5.3. Decoding with SVM-based binary detectors. (A) Three song sparrow syllables that show note composition. A and B are joined to form AB. (B) Onsets generated by the responses of feature detectors trained for A, B and AB when AB is given as input. (C) Responses of binary detectors flagged during recognition. (D) Binary detectors effectively select AB over A or B. The middle panel shows the prediction stream generated by feature detectors. The bottom panel shows the stream after the responses of binary detectors.

To 3.2%. The source of this, and the main difficulty we encountered, was correctly inferring the boundaries of notes inside syllables from the detectors’ onsets and offsets. This was particularly difficult in the regime up to 15 syllables per type. To circumvent these difficulties, we decided to look for a framework that simultaneously discovered the important features for classification and the same time trained binary detectors. We describe it in the following section.

5.1.5 Revisiting boosted binary detectors

Before we move to the next section, we briefly mention that our boosted binary detectors of Ch. 4 can also be used to distinguish syllables which share notes. However, they don’t automatically discover which syllables and notes those are. In Fig. 5.4 we illustrate this point using a Bengalese finch song snippet. The syllable labeled as A contains syllable B as its start. The mid-panel shows two pools of feature detectors trained for A and B. Note how the entire pool of B responds to both syllable types. Some, but not all, units in A respond to syllable B. A binary ensemble trained for A versus B using adaboost is shown in the bottom panel. Some units respond to other syllables during the song, but the entire ensemble responds to the end portion of A (which we denoted by $A \setminus B$ in the previous sections). After queried, binary detectors correctly select A as the prediction
Figure 5.4. Two-layer decision system for continuous recognition of syllables. The mid-panel shows feature detectors trained for syllables A and B. Syllable B is the start of A. Units trained for B respond robustly to A. The bottom panel shows how binary units trained for A versus B correctly predict the identity of the first syllable.

for the first syllable.

5.2 Recognition with temporal landmarks

In this section, we introduce changes to both the recognition and the reconstruction methods we’ve developed. We redesign our system so that it doesn’t rely on the auxiliary variables $\xi_i$ introduced in Ch. 3. This means that we treat onsets and offsets independently. We reintroduce the concept of anchor points and show how they can help with segmentation. Then, we show how onset/offset clustering can be used to implement pooling on delay lines and support fine-scale segmentation of syllables. This in turn helps to solve the temporal warping problem without the need of the auxiliary variables $\xi_i$. 
Additionally, pooling provides a systematic way to solve the composition problem.

5.2.1 Synchronous activity, detection, and reconstruction

With the delay grid of LIF neurons introduced in Sec. 3.3.1 we had a simple mechanism for temporally smearing spikes in order to support integration over a range of relative delay times. Now that we only have access to onsets and offset, we will attempt to reuse that idea. The LIF neuron uses an exponential kernel, but will use a Gaussian kernel instead:

\[ K(t) = \exp\{-t^2/2\sigma^2\}. \] (5.1)

First we need to estimate the kernel’s width \( \sigma \). It needs to be wide enough to produce a single peak for the preferred input sequence, but also narrow enough so that neighboring syllables don’t blend one into another. In addition, each template should have its own kernel, so we label the widths by \( \sigma_i \), where \( i \) indexes the template. We use the pool of trained templates to estimate the widths in the following way. First, for each syllable type, we loop through each template and use it to align the onsets and offsets of all other templates of the same type. This creates a sample of temporally aligned spikes for each template. These samples represent how well each template works as a delay line for their own syllable type. If the sample spreads out too much, we need to compensate for the spread by increasing the width of the kernel. If the sample is narrow, we can use a small width. To prevent too small or too large widths when the sample size is small, we estimate the kernel’s width with

\[ \sigma_i = \max\{t_{\text{min}}, Q^i_3 - Q^i_1\}, \] (5.2)

where \( Q^i_1 \) and \( Q^i_3 \) are the first and third quartiles of the spike time sample for template \( i \). The quantity \( t_{\text{min}} \) ensures that \( \sigma_i \) doesn’t go below one frame. In all our results we fix \( t_{\text{min}} = 3 \text{ ms} \). Note that, since we’re treating onsets and offsets independently, we need to estimate, for each template, two widths, \( \sigma^i_{\text{on}} \) and \( \sigma^i_{\text{off}} \). To keep the notation simple, we will hide the on/off indices. In Fig. 5.5A we show the kernels for onsets and offsets for two templates of two Bengalese finch syllables.

With the widths \( \sigma_i \) fixed, we can kernelize the onsets and offsets and use them in delay grids for detection. That is, for a sequence of onsets (or offsets) at times \( t_1, \ldots, t_N \), we compute:

\[ V_i(t) = \sum_{\tau=t_1}^{t_N} \exp\{-(t - \tau)^2/2\sigma^2_i\}, \] (5.3)
Figure 5.5. Gaussian kernels for detection with onsets and offsets. (A) The first column shows two Bengalese finch syllables. The second and third columns show the histograms of relative spike times after passing onsets (second column) and offsets (third column) through delay lines. Each histogram includes 14 syllables. Superposed in green and red we show Gaussian kernels estimated as the difference between the third and first quartiles of the histograms. (B) Anchor points in continuous recognition. The third panel shows onsets and offsets. The last panel shows onset and offsets after passing them through delay lines. The second panel shows the result of kernelizing and adding the aligned onsets and offsets (Eq. 5.3). The two traces effectively segment the syllables in the song. We call the peaks marked by vertical dashed lines anchor points.

where the index $i$ labels the template.

We now need to replace our reconstruction algorithm for one that uses only onsets and offsets, and not the sustained response between on and off (that is, the plateaus $\xi_i$ from Eq. 3.13). For that, recall that in Sec. 4.3.3.2 we introduced the concept of the anchor point. Anchor points are simply the peaks in the output trace of the delay line, $V_i$. This idea is not new. The major difference now is the kernel. But, in the next sections, we will extend it by including anchor points not just at the beginning and the end of syllables but also in the middle. This will also address our concern of Sec. 4.3.3.1 regarding pooling and help segment not only the song but the syllable itself. Notably, it will help with the problem of temporal warping. Since we treat onsets and offset independently, we will also allow anchor points to be created independently by both on and off sequences.

How do we use anchor points in practice? In Fig. 5.5B we show anchor points segmenting a canary syllable during continuous recognition. The third panel shows the sequences of onsets and offsets. In the last panel, the onsets and offsets are aligned by
a template. The second panel shows the kernelized responses, $V_i$, of onsets and offsets. The anchor points are illustrated by vertical dashed lines.

Reconstruction can be done reusing our discussion from Sec. 4.3.3.2 (see Fig. 4.9). We introduce two modifications to make reconstruction accurate. First, with the kernels now fixed, we use the templates again to estimate the distributions of relative anchor points. For a given onset anchor $a_{on}^i$ associated with template $i$, we generate a sample of $P(a_{off}^i|a_{on}^i)$ by passing the other templates through template $i$’s delay line. This distribution helps us find the end of the syllable when the beginning is found during decoding. Finally, we also collect relative spike time samples from $P(s_i|s_j)$ for each pair of detectors $i$ and $j$ in the templates. This idea is also recapitulated from Ch. 3, where we constructed spike bigrams.

Now we describe in detail the steps for reconstruction. Reconstruction start after detection, so we assume that the initial anchor point $a_{on}$ is given. To simplify the notation we hide the template index. From $a_{on}$, we use the symmetric interval of $[a_{on} - \Delta t/2, a_{on} + \Delta t/2]$ to locate the onset that is the closest to $a_{on}$. This is similar to what we illustrated in 4.9. In fact, we estimate $\Delta t$ in the same way we did in Sec. 4.3.3.2. Next, we use this onset to reconstruct the sequence of onsets sequentially. That is, from the current onset we look for next one in the sequence and repeat this process until we reach the end. To help with reconstruction at this point, we use the samples we obtained of the bigrams $P(s_i|s_j)$. Now, instead of estimating this distribution as we did in Ch. 3, we simply calculate the median absolute deviation (MAD) of the sample and fix the search interval to 5 times the MAD. With the onset reconstructed, we select the next anchor $a_{off}$ as the one that is the closest to the median of the sample of $P(a_{off}|a_{on})$ plus the value of $a_{on}$. Once $a_{off}$ has been found, we reconstruct the offset sequence in the same way we reconstructed the onset.

Anchor points help to segment and classifying syllables like the one shown in Fig. 5.5B. But this idea doesn’t perform as well for syllables that warp temporally because of the rigidity of the delay line. Consider, for instance, the syllable in Fig. 5.6. In the bottom, we show the on/off pattern after they have been passed through a delay line significantly shorter than the syllables being decoded. The first onsets align reasonably well, but the second one set creates multiple peaks. The offsets turn out poorly aligned as well. Our graph-based approach took care of this problem by allowing template onsets and offsets to match warped versions of themselves. Can we extend our discussion in this section to allow for time-varying syllables?
Figure 5.6. Delay line does not support time-varying patterns of onsets/offsets. The onsets and offsets shown were passed through a delay line significantly shorter than the two canary syllables depicted. Neither onsets nor offsets align well in the end.

5.2.2 On/off clustering and pooling

To understand why aligning, kernelizing and integrating onsets and offsets with a delay line works poorly for time-varying syllables, consider the artificial example of Fig. 5.7. This example shows that when a shorter template is aligned to an input, the output will be misaligned at the portions where the stretching occurs. This in turn generates the double peaks we observed in Fig. 5.6. To fix this problem, we propose that onsets and offsets should be pooled into smaller time-locked groups. To understand what this means, consider the four groups color-coded in Fig. 5.7B. These clusters show little in-group temporal variance, but are allowed to stretch arbitrarily as a single unit in order to support temporal warping of the syllable. In Fig. 5.7C we illustrate clustering of onsets and offsets of a Bengalese finch syllable using spectral clustering and 3 clusters.

What clustering effects to is pooling the delay line. This allows us to segment syllable using multiple anchor points. In Fig. 5.7D we show an example with 4 anchor points in a Bengalese finch syllable (that is, using 2 clusters for onsets and 2 for offsets). In the following section we describe segmentation in more detail.

How do we cluster spikes into groups representing synchronous activity? Here we try this idea using the normalized cuts algorithm of [81, 82]. For that, imagine that the onsets and offsets are generated independently by individual units. We can use the templates to look at the distributions of relative onset and offset times from pairs of
units after they have been through a delay line. For example, in Fig. 5.8A we show the aligned onsets from units 0 and 10 in a Bengalese finch syllable. In both cases there is little spread amongst the onsets of the two units. They are also very close to each other (for this syllable type we had about 79 trained detectors) so we would like to put them in the same cluster. To fix notation, suppose we have $N$ units and let us look at onsets first. First, to each of these units we assign a node in a fully connected graph. The edges connecting the nodes represent how synchronous each unit is in respect to all other units, as illustrated in Fig. 5.8B. Next we can construct a $N \times N$ matrix of weights $W_{on}$ representing synchronicity strength. Finally, we rearrange $W_{on}$ and put it in block representation. If we manage to do this last step, we can easily identify the clusters of synchronous units.

To implement this idea, we need to define precisely how to calculate the weights in $W_{on}$. In our earlier attempts, we tried using the inverse of the variance and the mutual information between onset/offset patterns such as the ones in Fig. 5.8A. The hope was that these quantities would measure the spread between the distributions of activities of units. However, a simpler approach is to use the fact that our sequences are already sorted. Our detector ids are monotonic functions of their preferred features inside the

Figure 5.7. Clustering onsets and offsets implements pooling. (A) On/off patterns do not align properly when the template is significantly shorter than the input. The output shows that the misalignment happens at the portions of stretching. (B) To solve the temporal warping problem, we cluster onsets and offsets into smaller groups that show little temporal distortion. The alignment is done by pooling the delay lines. (C) Examples of clustered onset and offset patterns from a Bengalese finch syllable. (D) Pooling the delay line generates multiple time-locked anchor points that allow us to segment the syllable into shorter chunks.
Figure 5.8. Spectral clustering for on/off patterns. (A) Sample of the distribution of onsets taken from two units in 5 templates of a Bengalese finch syllable. (B) We hypothesize that the onset times for some units vary less in respect to their neighbors’ than in respect to other units (right panel). We construct a graph with nodes representing units of onsets or offsets and the connections representing how synchronous each unit is in respect to its neighbors. Cliques with strong edges in this graph represent synchronous units, and we attempt to cluster them (left). (C) Connectivity matrices of onsets and offsets using a Gaussian kernel of width 3 ms. (D) Spectral clustering finds the blocks in the connectivity matrices and cluster onsets and offsets accordingly. Note how the offset clusters change significantly when the kernel width increases.

positive syllable. So a more sensible idea is to take into account the relative distance between units in the sequence. For that, we experiment with Gaussian kernels of widths ranging between 3 ms to 10 ms. In Fig. 5.8C we show the weight matrices $W_{on}$ and $W_{off}$ for the Bengalese finch syllable from panel 5.8A. Spectral clustering then finds the blocks these matrices independently to cluster onsets and offsets. In Fig. 5.8D we show 4 instances of the same Bengalese finch syllable. We also show their respective sequences of clustered onsets/offsets after using spectral clustering with Gaussian kernels of widths 3 ms and 6 ms. Note how the widening of the kernel keeps the onset clusters relatively similar but drastically changes the offsets.

5.2.3 Pooling and composition

We now return to the main focus of this chapter and briefly discuss in more detail how pooling can help in the composition problem. The number of clusters isn’t fixed, but we can select it upon visual inspection of the syllable. For a syllable with 2 notes, a reasonable choice is 2 clusters. Each template then defines 4 anchor points, which can
Figure 5.9. Pooling for note composition in the song sparrow. (A) and (B) Anchor points in response to 4 song sparrow syllables. Anchor point color indicates the cluster identity. In total, there are 5 templates and 4 clusters per template, resulting in 20 anchor points in total. (C) and (D) Segmented chunks of the syllables in A and B by anchor points generated from 4 different templates. The same syllable is used in each segmentation.

be taken in pairs of onsets and offsets to segment the syllable in two. In Fig. 5.9 we show two examples for the song sparrow. In panels A and B, we show the responses of 5 templates, each containing 4 anchor points. Below, in panels C and D, we show the segments extracted from two syllables (one in each panel) by 4 distinct anchor points of each syllable type. As the anchor sequences shown in panels A and B suggest, the segments in C and D are very similar.

5.3 Conclusions

We discussed the problem of compositionality in birdsong syllables and how it relates to automatic recognition. Our discussion focused on the song sparrow’s song, but it also drew examples from Bengalese finch syllables. The chapter was divided in two part. First, we saw how feature detectors can be used for automatic discover of subunits in syllables. We also discussed training binary detectors to solve the recognition task when syllables share notes. In the second part, we introduced new ideas in an attempt to solve both the temporal warping and the compositionality problems simultaneously. We developed the concept of pooling and implemented it using clustering. We then discussed
how pooling can be used for recognition and for syllable and song segmentation.

In going forward, we envision a spike-based recognition system that uses pooling and performs segmentation based on anchor points. The short segments can then be analyzed more carefully by binary detectors or other specialized units. The anchor points are also precise and can be used as references for our method of sequence completion introduced in Sec. 3.3.5.2.

The techniques we developed in this chapter may also be useful in speech recognition, where words are composed of consonants and vowels. By adapting the clustering algorithm, we expect that pooling may be able to segment words into their phonetic subunits. Feature detectors and binary detectors may be trained to discriminate words based on these subunits.
Chapter 6
Reconstruction of neuronal somata from slice imaging

Reconstruction of neuronal morphology is important for understanding the spatial organization and electrical properties of neurons. Much research effort has been undertaken in developing tools for accurate reconstruction of complex dendritic arbors. Less attention has been given to the problem of reconstruction of somata. While the dendritic cable is typically the major contributor to the neuron’s total surface area and total capacitance, the soma can also represent a significant fraction of the neuron’s area and volume. Thus, there is a need for specialized and simple tools to help reconstruct the surface of somata. In this chapter, we briefly describe a simple computational method we have developed for reconstructing neuronal somata using slice imaging.

6.1 Soma morphology

Neuronal morphology can be reconstructed from slice imaging. A thin tissue slice is fixated on a surface and scanned along the surface plane at a set focal depth. The scanning process is repeated for multiple depths to cover the entire volume of the slice. The end product is a stack of images representing the volume of tissue (Fig. 6.1A). The researcher then uses the image stack to trace the volume occupied by the neuron, thus reconstructing its morphology.

A number of excellent tools for neuronal reconstruction from bright-field microscope images have been built. The most commonly used data structure for representing the reconstruction is the SWC format. The SWC file is a linked list, where each element has an id, a parent id, the coordinates of a point in space, a radius value and a tag. The SWC structure can be pictured as a cascade of linked spheres descending from a single...
Figure 6.1. Measuring somatic area. A. A z-stack represent a 3D object as a sequence of cross-sections. The object’s surface area can be reconstructed from the contours in all planes. B. Annotation schema commonly used to represent somata include spherical approximation (upper right), multiple overlapping spheres (bottom left), and single-plane contour (bottom right). Allen Cell Types Database, celltypes.brain-map.org. Layer 5 V1 neuron in the mouse, celltypes.brain-map.org/experiment/morphology/475586130. Upper right reconstruction from the Allen Cell Types Database. Bottom left and right reconstructions are modifications of the upper right. Soma in blue, dendrite in green and axon in gold. C. Morphology repositories such as NeuroMorpho.Org includes multiple annotation schema and distinct definitions of soma surface area, which leads to multimodal distributions [83].

root element. The tag’s value is set to indicate whether the point is part of the soma, the dendrite or the axon. The SWC is a smart choice of data structure for representing neuronal processes, as most neurons are formed by a cell body with dendrites and an axon branching out in space. Dendrites and axons are typically thin enough to be well represented by SWC points. The real physical shapes of dendrite and axon cables can be approximately well recovered with trapezoidal cylinders connecting every pair of adjacent SWC point.

Somata come in different shapes and sizes and may not be well approximated by a collection of linked spheres. To contour this difficulty while keeping the SWC format, researchers have adopted a variety of annotation procedures. These include single spheres, multiple overlapping spheres and single-plane contours (Fig. 6.1B). In many instances, researchers adopt the radius of the disk inscribed in the maximally projecting section plane of the image stack. But there is no set standard shared by reconstruction databases.

Related to this issue, there are also conflicting reports in the literature regarding the definition of the surface area of the soma. The maximal projection area of the image stack is often labeled as surface area. Failure in making this distinction can lead to substantial
differences in the surface area. For spherical somata, for instance, this difference amounts to a factor of 4.

Over time, different definitions for the soma surface area accumulate to yield mixed statistics. It reflects, for example, in multimodal distributions of surface area found in repositories of morphology reconstruction (Fig. 6.1C). Crucially, these estimates bias the representative somatic area and may harm modelling efforts where the precise value of capacitance per compartment is important.

Other tools have been proposed for automatic detection and reconstruction of somata [84, 85, 86]. Our focus here differs in that we are interested in fast and accurate estimation of surface area. We provide a simple recipe for annotation and reconstruction of somata. We use it to reconstruct the surfaces of 30 somata in the mouse’s V1, and we compare our results to existing reports.

6.2 Reconstruction using surface triangulation

We have formulated a simple procedure for reconstructing the surface of an object using manual annotations of slice images as input. Here we describe the steps of the reconstruction process using a soma in the mouse’s V1 as example. We also describe benchmarks with objects of known shapes and apply our method to reconstruct 30 somata from the mouse’s V1.

The soma of a neuron in a high quality Z-stack can be easily delineate with specialized software. Here we used ShuTu [87] to manually mark the pixels occupied by the soma in each slice image. For each image stack we fixed the contrast and brightness values such that the outline around the soma was sharp and clear in the maximally projected slice plane. For planes distant from the maximal projections, where the image typically blurs the edges around the object’s cross-section, we drew the masks around pixels with very high contrast only. We export these annotations as binary masks representing the total volume occupied by the soma (first two rows of Fig. 6.2A).

The region containing the soma needs to be simply connected for our method to work. Before tracing the contours in each binary mask, we apply morphological closing with a 3x3 structuring element. Closing ensures that small gaps are absent from the connected regions. Next, we apply dilation to the mask and subtract it from itself. This operation results in a pixel-wise thin contour. Next, we trace a continuous thin contour around the soma in each binary mask (third row of Fig. 6.2A). At this stage, because all contours may not have the same length, we traverse each contour counterclockwise and partition
it into 32 segments of approximately equal number of pixels (last row in Fig. 6.2A; color represents segment label). Finally, we polygonize the contours by retaining only the first and last points of each segment so the number of edges in each polygon is equal to the number of segments (Fig. 6.2B). To remove potential kinks and rough edges on the surface, we also smooth the reconstruction along the axis orthogonal to the image plane by applying a 3-point average filter 4 times.

Once we have converted the slice images into polygons with an equal number of edges, we construct surface patches to estimate the total surface area. The polygons are stacked to represent the original order of the image slices. We connect the vertices of each adjacent slice according to their segmentation labels. This step results in a sequence of quadrilaterals covering the space between the two slices (Fig. 6.2C). Thus,
each consecutive pair of polygons represent a patch of lateral surface area of the soma. We divide the quadrilaterals in two to triangulate the surface patch. The total surface area of the soma is the sum of the areas of the triangles plus the areas of the bottom and top caps (Fig. 6.2D).

### 6.2.1 Reconstruction of known geometrical shapes

To evaluate our method’s accuracy, we have reconstructed the surfaces of simple objects with known areas. First we generated binary masks from the cross-sections of each object at varying depths along the vertical axis (Fig. 6.3A). We then found the surface area of the objects with our triangulation method. For a sphere, we found that the relative error to the sphere’s true surface area decreased rapidly with the number edges in the polygonal contours. At 15 edges per slice, the relative error was less than 1% (Fig. 6.3B). Other shapes positioned at varied orientation angles in respect to the slice’s plane yielded similar decreasing errors. We note that our method also accurately reconstructs concave objects, such as the gourd-like shape of Fig. 6.3, so long each slice projection is a simply connected region.

### 6.2.2 Reconstruction of somata in the mouse V1

We retrieve image stacks and reconstruction metadata of 13 neurons from the Allen Cell Types Database (http://celltypes.brain-map.org/) using the Allen Brain Atlas API (http://help.brain-map.org/display/api/Allen+Brain+Atlas+API). We also obtained the images of 17 neurons from the Allen Reconstruction Challenge. All neurons used here were located in the mouse’s V1. To reduce the sizes of the images, we clipped a small
patch around the soma in each stack. We then reconstructed the somata of these 30 neurons from the mouse’s V1. The images consist of thin slice sections of biocytin-filled neurons.

We found a variety of sizes and shapes, many of which resembled spheroids (19 selected reconstructions shown in Fig. 6.4A). We found a distribution of surface areas concentrated around $534 \pm 39 \mu m^2$ with a longer tail towards larger values (Fig. 6.4B). Our distribution of surface areas differed from the ones from Neurmorpho.org and the Allen Cell Types Database in that it does not concentrate any density below $250\mu m^2$. Direct comparison of a subsample of 13 somata with the reported values from the Allen Cell Types Database indicates that our estimates yield slightly larger areas for somata larger than $250\mu m^2$ and much larger areas for those below $250\mu m^2$ (Fig. 6.4C). We also found that our sample of 30 somata surface areas resembles in average that of an spheroidal shell with semi-minor axis defined by the maximally projecting surface (Fig. 6.4D). A direct fit shows that the surface area of the somata increases at a rate of approximately 5 with the maximally projecting surface area. Thus, in average, a prolate spheroidal approximation is justified for somata in the mouse’s V1. However, we note that some may be more consistent with an oblate spheroid or a sphere.

6.3 Conclusions

We described a simple method for accurate reconstruction of the surface of somata using manual annotations. Manual annotations of Z-stacks can consume time to produce, particularly when the neuron spans many slices or the image quality is poor. In our method, only the contour surrounding the soma needs to be drawn on top of each slice. We showed that it reproduces the surface areas of objects of known shapes. We also reconstructed the somata of 30 neurons in the mouse’s V1. We found that their surface areas concentrate around $534\mu m^2$, and we found no soma below $250\mu m^2$. 
Figure 6.4. Reconstruction of somata in the mouse V1. A. Selected reconstructions of 19 somata. B. Distribution of surface areas and comparison with the distributions found in two databases. Note the difference in density for small surface areas. C. Direct comparison of surface areas of 13 somata with the reported values show that the values found at the lower end may have been underestimated. D. The shape of the somata in the mouse V1. Most somata are consistent with a prolate spheroid, although some are more consistent with oblate spheroids or spheres. Least-squares fit yields a slope of 4.99. Red line with a slope of 4 represents the sphere.
Bibliography


[53] Nima Mesgarani et al. “www.sciencemag.org Downloaded from on December 30, 2014 www.sciencemag.org Downloaded from on December 30, 2014 www.sciencemag.org Downloaded from on December 30”. In: (2014), pp. 1–6.


Vita
Leonardo Estêvão Schendes Tavares

Education

Ph.D. in Physics 2014–2022
The Pennsylvania State University, University Park, PA

M.Sc. in Physics 2012–2014
University of São Paulo (USP), São Paulo, Brazil

B.Sc. in Physics 2008–2012
University of São Paulo (USP), São Paulo, Brazil

Awards

The Pennsylvania State University, University Park, PA

Graduate Teaching Assistant Award
Downsbrough Graduate Fellowship in Physics