A Fast and Accurate Zebra Finch Syllable Detector

Ben Pearre[1,¤], L. Nathan Perkins[1], Jeffrey E. Markowitz[1], Timothy J. Gardner[1]

[1] Department of Biology, Boston University, Boston, Massachusetts, United States of America

¤ Corresponding author: bwpearre@gmail.com (BP)

Abstract

The song of the zebra finch is strikingly consistent. Efforts to understand motor output, pattern generation, and learning have taken advantage of this consistency by investigating the bird's ability to modify specific parts of song under external cues, and by examining timing relationships between neural activity and vocal output. Such experiments require that specific moments during song be identified as the bird sings. Various syllable-detection methods exist, but many require special hardware, software, and know-how, and details on their implementation and performance are scarce. We present an accurate, versatile, and fast syllable detector that can control hardware at precisely timed moments during zebra finch song. Most moments during song can be isolated and detected with $> 95\%$ accuracy, easier syllables can be detected 99.5% of the time, with fewer than 0.1% of songs producing false positives. The detector can run on a stock Mac Mini with a triggering delay around 3 milliseconds and trigger jitter with $\sigma \approx 1.2$ milliseconds.

1 Introduction

The adult zebra finch (*Taeniopygia guttata*) sings a song made up of 2-6 syllables, with longer songs taking on the order of a second. The song may be repeated hundreds of times per day, and is almost identical each time. This consistency presents a unique opportunity to study the neural basis of learning, audition, and control: we do not know of another model in which neural activity may be so easily correlated with motor output.

Quantify? Remove vacuous handwaving? Figure?

Why might anyone care about syllable detection? Insert a brief review of some of the coolest research done with song-triggered recording/stim. Perhaps at a minimum (a) vocal shifting and (b) song-aligned spike raster?

In order to take advantage of this consistency, it is useful to be able to detect selected moments during song, in order to align data or trigger other systems. Important considerations for such a detector are as follows:

Accuracy: How often does the system produce false positives or false negatives?

Latency: The average delay between the target syllable being sung and the detection.

Jitter: The amount that latency changes from instance to instance of song.

Versatility: Is detection possible on "difficult" syllables?

Ease of use: How much hand-tuning is involved? What level of expertise is required in order to produce good results?

Cost: What are the hardware and software requirements?

A variety of syllable-triggering systems have been used, but few have been described or characterised in detail. In 1999, [1] used groups of IIR filters with hand-tuned logical operators. Their system had a latency of 50 or 100 milliseconds (ms), and they do not report on jitter or accuracy. As accesss to computational resources became less expensive, approaches have changed: in 2009, [2] still uses hand-tuned filters, but now runs them on a Tucker-Davis Technologies digital signal processor. They report a latency of around 4 ms, but as with other filterbank techniques, it is not strictly a syllable detector but rather a pitch detector, and thus requires careful selection of target syllables. Furthermore, the method is neither inexpensive nor, based on our experience with a similar technique, accurate. [3] applies a neural network to a spectral image of song, and reports a latency of 4.3 ms. Further implementation and performance details are not available, and they report that their system is too deeply integrated into their experimental apparatus to be generally useful [personal communication]. In 2013, [4] describes a system that matches spectral images of template syllables using a correlation coefficient. With a fast desktop (Intel i7 six-core) running Linux and sporting a National Instruments data acquisition card, they report a hardware-only (no syllable-matching computations) latency and jitter of just a few microseconds, and the computation they perform should not much increase that. Drawbacks are some little hand-tuning, and false-negative rates around 4-7% (zebra finches and Bengalese finches, respectively) on a small dataset. In much other work, an allusion is made to a syllable detector, but no further information is provided.

Tim: that's my understanding of your conversation with Richard. Is this problem worth mentioning?

We developed a standalone detector based on the song spectrogram. Given a set of aligned songs, our software computes spectrograms and trains a neural network to output a TTL pulse at the chosen moment. The approach consists of three steps:

1. Align songs. The technique has been published in [5].
2. Choose one or more desired trigger syllables, and train a neural network to recognise them. This step is carried out offline. While any neural network software would produce equivalent results, we used Matlab 2015b, and we recommend a computer with at least 32GB of RAM.
3. Once trained and saved, the neural network is used by a realtime detection program that listens to an audio signal and indicates detection of the target syllables via a TTL pulse. We present three implementations that trade off hardware requirements, ease of maintenance, and performance.

Terminology: "syllable" is a unit, but this detector triggers on "song moments" or something?

This method makes the following contributions:

- Fast: latencies in the 3-6 ms range, with jitter around 1.5 ms.
- Accurate: false positive rates under 0.1%, and false negative rates well under 1%, for a variety of syllables. Balancing these rates depends on a single relative-cost parameter.
- Minimal hand-tuning.
- Runs on inexpensive hardware.
- Described in some detail here, with reference implementations provided.

We present the method in Section 2. Section 2.3 describes how we define and test performance. Section 3 presents our measurements. Section 4 gives an example usage case of the detector. We conclude in Section 5, and point to software resources in Appendix A.

2 Materials and Methods

2.1 Learning a detector

We begin with a few hundred time-aligned recordings of a given bird's song.

The spectrogram is computed at regular intervals—every 1-5 milliseconds, which we refer to as the frame time t_f.

One or more moments during the song must be chosen. Our interface presents the time-aligned spectrogram averaged over training songs, and requires manual input of the target times. The user may next tune the region in frequency and time used for syllable identification. Then we assemble the training set from the song data, train the network, compute optimal output unit thresholds, and save the network file and an audio test file.

Recognition region The neural network uses a contiguous set of the most recent timesteps (frames) from the song spectrogram. The frequencies F and duration T of this recognition region should be chosen in order to contain unique features of the target syllable and surrounding areas. This process could perhaps be automated, but we have not done so, as hand-choosing is neither difficult nor particularly error-prone.

Song micro-realignment As conditions change, and especially during undirected song, syllable length and relative timing may vary slightly, which introduces variations in the precise timing of each syllable. The alignment software we use ensures that songs are aligned at the point midway through the song, but if the target syllable is not at that point, it is helpful to re-align the songs at the point of interest. This may be accomplished by looking for peaks in the correlation of the time-domain signal with the song whose spectrogram is closest to average over the training set.

I hope that's the right thing to do. Seems to work, anyway...

Normalisation To accommodate differences in amplitude, due to changes in the relative position between the microphone and bird or due to other minor variations in recording, normalisation ensures that the detector is sensitive only to the relative spectral content. We found that a two-step normalisation works. The spectrogram in the time and frequency windows, in decibels, is normalised to have mean zero and unit standard deviation. This eliminates relative differences in song amplitude due to microphone position or line levels. Next, each time-frequency bin is normalised based on the training set, such that that bin has mean zero and unit variance. This scales each time-frequency bin based on the amplitudes seen in the training set. These two normalisation steps provide a set of inputs that were more robust to outliers and less likely to produce false positives during silence when evaluated against other normalisation schemes, such as linear or L2 normalisation.

What data is each norm computed over? Why does this do more than just the second step?

Over what set of measurements?

Over what set of measurements? This seems clearer than step 1, but the description of step 1's result sounds like what this would do.

Would it be worth putting in data to this effect?

Building the training set The neural network's training set is created in the typical fashion: the rectangular $|F| \times |T|$ recognition region in the spectrogram is simply reshaped into a vector of length $|F||T|$. The frequency range is constant, and every possible time interval of length $|T|$ is converted into a training input vector.

Training targets are, roughly, 1 if the input vector comes from the target time, 0 otherwise, for each target syllable. Since the song realignment may not be perfect, due to sample aliasing, and because the song spectrogram appears not to vary faster than the frame rate we chose, a strict binary target may ask the network to learn that practically identical samples should have opposite targets. Thus it is preferable to spread the output vector in time, such that at the target moment it is 1, and at

neighbouring moments it is nonzero. We found that a Gaussian smoothing kernel around the target time with $\sigma \simeq 3$ms serves well.

With inputs well outside the space on which a neural network has been trained, its outputs will be essentially random. In order to reduce the false positive rate it is useful to provide negative training examples that include silence, cage noise, non-song vocalisations, and perhaps songs from other birds. We have found that training with as low as a 1:1 ratio of non-song to song yields excellent results, although this will depend on the makeup of the non-song data.

Training the network The network is trained using Matlab's neural network toolbox. We tried a variety of feedforward neural network geometries, from simple 1-layer perceptrons to more complicated forms and many hidden nodes. Perhaps surprisingly, even the former yields excellent results on many syllables, but a 2-layer perceptron with a very small hidden layer—just 1 or 2 more units than the number of target syllables—was a good compromise between accuracy and training speed. Various other neural network geometries could be tried, as well as any other classifier that executes quickly. For more variable songs, deep structure-preserving networks may be more appropriate, but they are slow to train and unnecessary for zebra finch song.

Didn't you guys try reducing input size with an autoencoder? Wasn't that effective?

Matlab's neural network toolbox defaults to Levenburg-Marquardt training. This is a fast algorithm, but is memory-intensive, so multiple output syllables or high FFT frame rates require a large amout of RAM and increase training time to hours. Other training algorithms that use less RAM are much slower, and by default they will often terminate before converging due to their performance gradient going to 0.

Computing optimal output thresholds When the network is trained, outputs of the classifier for any input are now available, and will be in the interval (0, 1). We must choose a threshold above which the output is considered a positive detection. Finding the optimal threshold requires two choices. The first is the relative cost of false negatives to false positives, C. The second is the acceptable time interval: if the true event occurs at time t, and the detector triggers at any time $t \pm \Delta t$, then it is considered a correct detection. Then the optimal detection threshold τ is the one that minimises [false positives] $+ C \cdot$ [false negatives] over the training set, using the definitions of false positives and negatives given in Section 2.3. Since large portions of the cost function are flat, random-restart hillclimbing would be effective, but a brute-force search requires fractions of a second. For the results presented here, we have used $C = 1$ and $\Delta t = 20$ms.

I chose this large value for Δt before I did target syllable realignment, and it could probably be much reduced now. Does this number make our results look bad? Should I explain it, change it and rerun, etc?

Our parameter choices We used an FFT of size 128 or 256; a Hamming window; and chose a spectrogram frame time of $t_f = 1.5$ milliseconds. We usually define the network's input space to be 20-80 ms long, and to span frequencies from about 1.5-7 kHz, which contains the fundamentals and several overtones of most zebra finch vocalisations.

We found these parameters to work well across a variety of target syllables, but various other parameter sets yield results similar to those presented here. Some of the parameters trade off detection accuracy or detection timing vs. training time. For example, detection latency and jitter cannot be less than the frame rate, but increasing the frame rate increases the size of the training set and thus the training resources required and the detection latency.

2.2 Realtime detection

The architecture of the realtime detector requires that the most recent $|T|/t_f$ spectrograms be fed to the neural network at every timestep (1.5 ms in our example). Audio samples from the microphone are appended to a circular buffer. Every t_f seconds a new spectrogram is calculated by applying the Hamming window to the contents of the buffer, performing an FFT, and extracting the power. Outputs of the spectrogram, from the target frequency band, are appended to a second circular buffer. The spectrograms are sent to a static implementation of the previously trained neural network.

We tested three implementations of the realtime detector. For all of these tests, we ran the detector processes under the operating systems' default schedulers and process priorities, running typical operating system daemon processes but no user loads. The computers had ample memory resources.

Swift This detector uses the Swift programming language and CoreAudio interface for Apple's operating systems.

The Core Audio frameworks provide an adjustable buffer size for reading from and writing to audio hardware. Tuning this buffer size provides a tradeoff between the jitter in the detection and the processor usage needed to run the detector. We settled on a buffer size of 32 samples (0.7ms at 44.1kHz), as this created minimal system load while achieving detection within the desired lag and jitter.

Vector operations—applying the Hamming window, the FFT, input normalisation, matrix multiplication, and the neural network's transfer functions are performed using the Accelerate framework (vDSP and vecLib), which use modern vector-oriented processor instructions to perform calculations.

When the neural network detects a match, it instructs the computer to generate a TTL pulse that can be used to trigger downstream hardware. This pulse can be either written to the computer's audio output buffer (again, in 32-sample or 0.7-ms chunks) or sent to a microcontroller (Arduino) via a USB serial interface. Sending the trigger pulse via the serial interface and microcontroller is noticeably faster (2.2 ms lower latency), likely due to the fact that the audio buffer goes through hardware mixing and filtering prior to output.

The above code can be run on multiple channels of audio on consumer hardware (such as a 2015 Mac Mini) with little impact on CPU usage ($< 15\%$). Depending on the experimental needs, latency can be further decreased (at the expense of processor usage) by adjusting the audio buffer sizes.

LabView This implementation requires special software and hardware: LabView from National Instruments, and a data acquisition card—we use the National Instruments PCI-6251 card on a PC with an Intel Core i5-4590 processor at 3.7GHz (a relatively low-end machine) with 32 gigabytes of RAM, running Microsoft Windows 8.1 Pro.

This implementation has several drawbacks: due to the programming language it is difficult to modify and debug, and it requires Windows. However, our test configuration achieved excellent performance, and further gains should be possible if the implementation were retargeted onto FPGA hardware, or run on a faster desktop system.

Matlab This detector uses the built-in audio input and output jacks on a compatible computer, requiring no special hardware. We tested on a 2015 Mac Mini and 2015 Mac Pro. The Mac Pro does not have an audio input jack, so input was

through an M-Audio MobilePre external USB audio interface. Despite the faster processor, the latter system did not achieve higher performance than the former, perhaps due to USB data transfer overhead.

The most versatile implementation, Matlab runs on a variety of hardware and operating systems, and is the easiest to modify. While it did not perform as well on our test system as the other implementations, the convenience may outweigh the slight timing performance penalty for some experiments. Key to minimising jitter is the size of the audio buffer: on a 2015 Mac Pro or Mac Mini running Matlab 2015B the smallest buffer size that did not result in read underruns was about 4ms.

2.3 Quantification

Ground truth is given on the training set, and can be measured by presenting the recorded training songs as well as the canonical detection events. To this end, besides the trained network object, our learning code produces an audio file consisting of all of the training data on the left audio channel and a delta function at each moment of target syllable presentation on the right channel. Thus, when played on any audio player, the left channel may be provided as input to the detector, and the the detector's detection pulses may be compared against the right channel.

Accuracy The Matlab neural network toolbox breaks the given training set into three groups: data on which the network is trained, data used to validate the progress of the training algorithm, and holdout test data used only as a final measure of performance. We further withhold a portion of the training data in order to provide another evaluation of performance on unseen data.

It's a little more complex than that (cross-validation), but can we go with this?

But another test set layer is just paranoia!

We define the accuracy of the network based on its classification performance per frame. In order to avoid the apparent problem of every non-detected non-syllable counting as a true negative, we also tried defining accuracy on a per-song basis, such that a song without the target syllable counted as a single true negative. Computing the optimal output thresholds on a per-frame basis resulted in higher thresholds and thus a lower false-positive rate, with minimal consequences to the false-negative rate, while also providing a valid definition of false-positive rate for data streams that had not been segmented into songs.

The accuracy as defined above is used for computing the optimal thresholds above which the network's output should be interpreted as a match on the training data as described in Section 2.1, for evaluation of the detectors on the training songs, and while live.

We present the resulting confusion matrix for a few sample songs, and, for simplicity, we present a summary of detector accuracy using the area under the ROC curve.

Timing We evaluate the time taken from the presentation of the target syllable to the firing of the detector's TTL pulse. While playing the audio test file from another device (such as a mobile phone), the TTL output from the ground-truth channel of the audio output may be used as the trigger pulse for an oscilloscope, and compared to the TTL pulse produced by the Swift detector, which sees only the birdsong channel of the audio file. For this purpose we used a pulse generator (Philips PM 5715, with a listed latency of 50ns, jitter of $\leq 0.1\%$ or 50ps, whichever is greater) to widen the detector's output spike to a number much larger than the jitter (100ms). This obviates pulse length jitter in the output device by essentially discarding the falling edge of the output pulse. The oscilloscope is then set to averaging mode (128-trigger average) in order to collect timing data. The canonical signal is the trigger at $t = 0$,

and the average of the detector's detection events will be seen as a low-to-high transition with form approximating the cumulative probability distribution function (CDF) of the detector's output in response to the chosen song event.

Mean latency is then given as the halfway point of that detection sigmoid. It is a helpful number, but not a critical one, since a detector with high but constant latency can be trained to trigger at a point somewhat before the true moment of interest. Often more important is latency jitter: how much random variability is there in the latency?

We obtain both of these numbers by performing a maximum-likelihood fit of a Gaussian distribution to the timing data obtained from the oscilloscope.

It is possible to calculate the ideal latency and jitter, which reflects the inherent latency and jitter in the time shift used in calculating new columns of the spectrogram. By passing a recorded audio sequence into the detector and assuming no input or output latency, it is possible compare the how many additional audio samples beyond the syllable are needed before the spectrogram matches the inputs needed to trigger the neural network. Given the detector used for benchmarking the LabView and Swift implementations of the detector, the ideal implementation will have a latency of $\mu = 1.1$ms; $\sigma = 1.0$ ms. This latency reflects the FFT size used for calculating the spectrogram, the FFT time shift between columns in the spectrogram and the width of the Gaussian smoothing kernel applied to the ground truth data when training the neural network.

3 Results

Fig. 1 shows a typical song, with six target moments selected. We trained the network with 8 hidden units for the 6 detection targets spaced 10ms apart. Matlab's self-report of detection performance for each iteration of the song is shown in Fig. 2, with ROC curves shown in Fig. 3. The beginning of the syllable is difficult to detect, with a few detection events considerably earlier than the correct moment. But the syllable quickly becomes reliably identifiable. By the time the detector has seen 50ms of the syllable—the sixth detection point—performance is good: the area under the ROC curve is 0.992.

I should generate, say, 100 random detection points in a few songs, so I can say what something like "average accuracy" is.

Accuracy for a typical good syllable is identical across the three detectors. For one of our examples of a well-chosen syllable from a random bird, the confusion matrix was as follows:

Should be! Verify! And/or explain that there is a bug in the LabView implementation, if there in fact is. ALSO need to present a picture of the test syllable or etc...

	True	
	pos	neg
Detected pos	632	1 (0.07%)
neg	4 (0.63%)	1271

Fig. 4 shows the latency and jitter for the three implementations of our detector running on a single syllable from another bird, using the test audio file generated during training on that song (not shown). The maximum-likelihood estimates of the mean μ and standard deviation σ of the detector latencies on this training set are as follows:

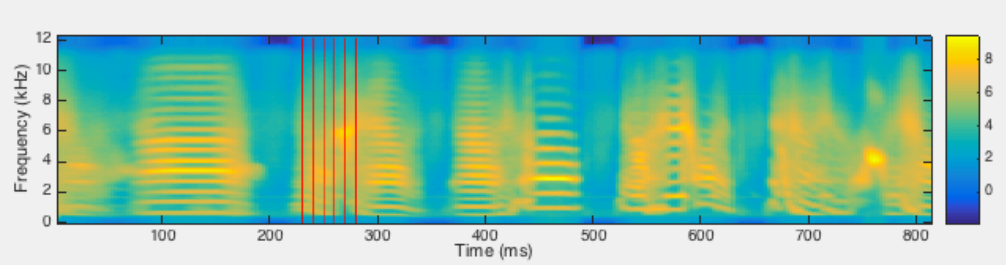

Figure 1. A spectrogram made by averaging over 526 songs. The 6 target syllables, spaced 10ms apart, are marked by red lines.

	Latency (ms)	
Detector	μ	σ
Ideal	1.1	1.0
Swift+Arduino	2.8	1.2
LabView	3.2	1.4
Matlab+Arduino	6.7	1.3
Swift+audio	6.8	1.3
Matlab+audio	16	2.9

4 Case Study?

Do we want to include a case study? I think the paper is on the long side already, but it might be interesting and let Jeff earn that coauthorship. I believe the work using this detector just got submitted and won't be published for a while, right?

Jeff is using the detector for something cool, and it might be fun to get him to describe it in a paragraph or two.

Failing that, I can immediately do song-aligned X spike rasters if lw95rhp is still willing to sing... but I have not heard him do so lately.

5 Discussion

This syllable detector is appropriate for zebra finch song, and although we did not test in other species, it is likely to work well for Bengalese finches and probably other species. It offers the following benefits:

- The detector is accurate. False positive and false negative rates can both be well under 1%, and trading these two numbers off against each other is through a single relative-cost parameter.
- Latency can be around 3 ± 1.2 ms or better depending on hardware.
- Works on a wide range of target syllables.
- A single detector can easily generate different target pulses for multiple syllables.
- Requires minimal hand-tuning.
- Runs in realtime on inexpensive consumer-grade hardware.
- We recommend training the network on a computer with at least 32GB of RAM.

As described, the detector only identifies matches that are similar to the training syllables. A common experimental paradigm requires detecting the frequency of syllables. This can be accomplished by taking an autocorrelation of the harmonic stack at the moment of interest and finding peaks, which will yield the fundamental

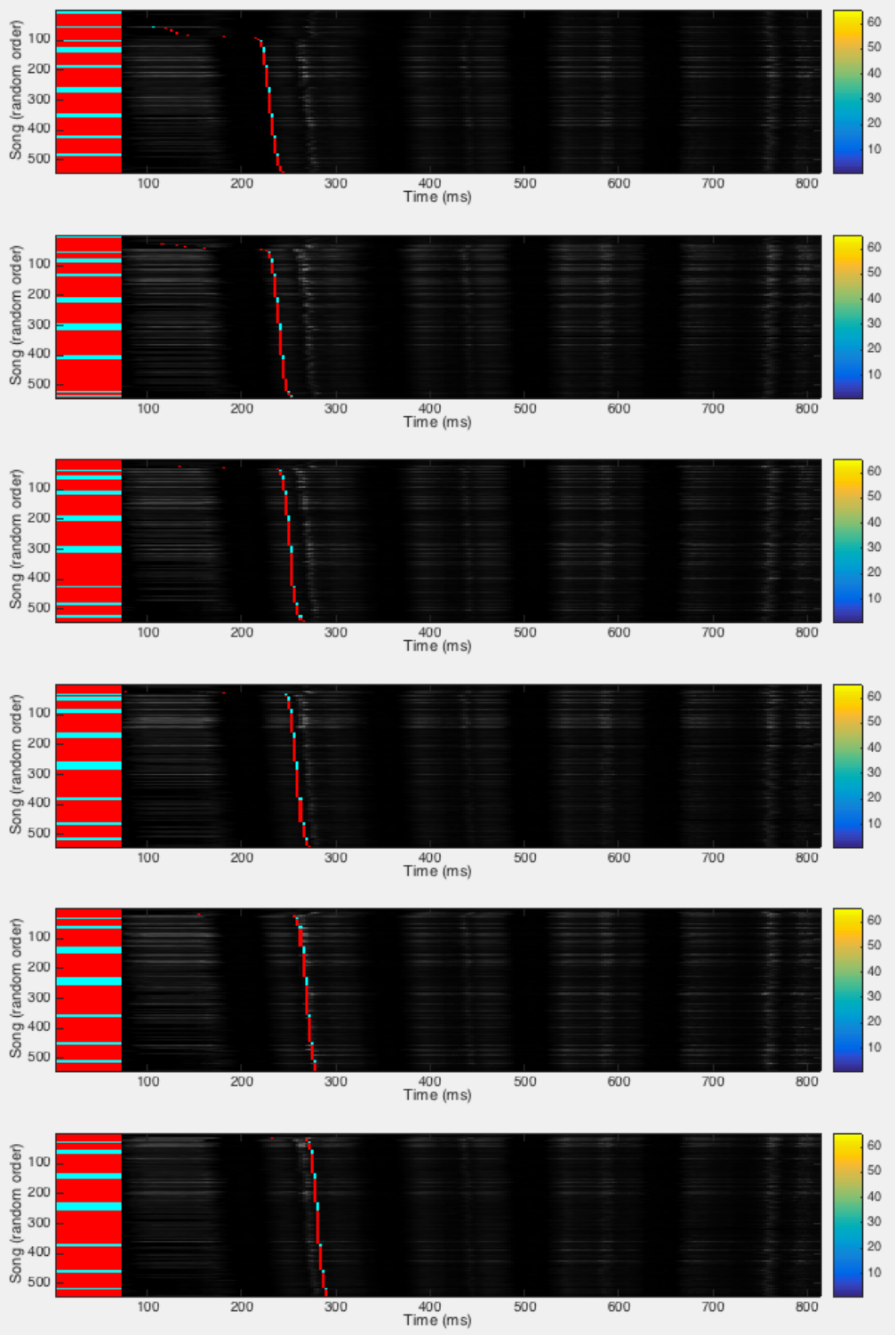

Figure 2. Detection results. Each plot shows detection events for the corresponding syllable as shown in Fig. 1. The horizontal axis is time, and the vertical axis is the index of the 526 single song presentations. The grey shading shows the total audio energy of song Y at time X. The bar on the left is the same width as the detection window, so no detection events can happen within that region, and its colour code shows training songs to the right of red regions and unseen test songs to the right of cyan regions. For visualisation of the distribution, songs have been stably sorted by the time of detection events.

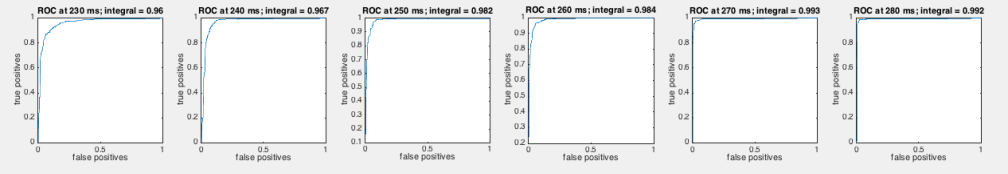

Figure 3. ROC curves for the detection of targets shown in Figure 1. The first one, at the beginning of a syllable, is the most difficult to detect, and as more structure emerges in the current syllable, accuracy approaches 100%.

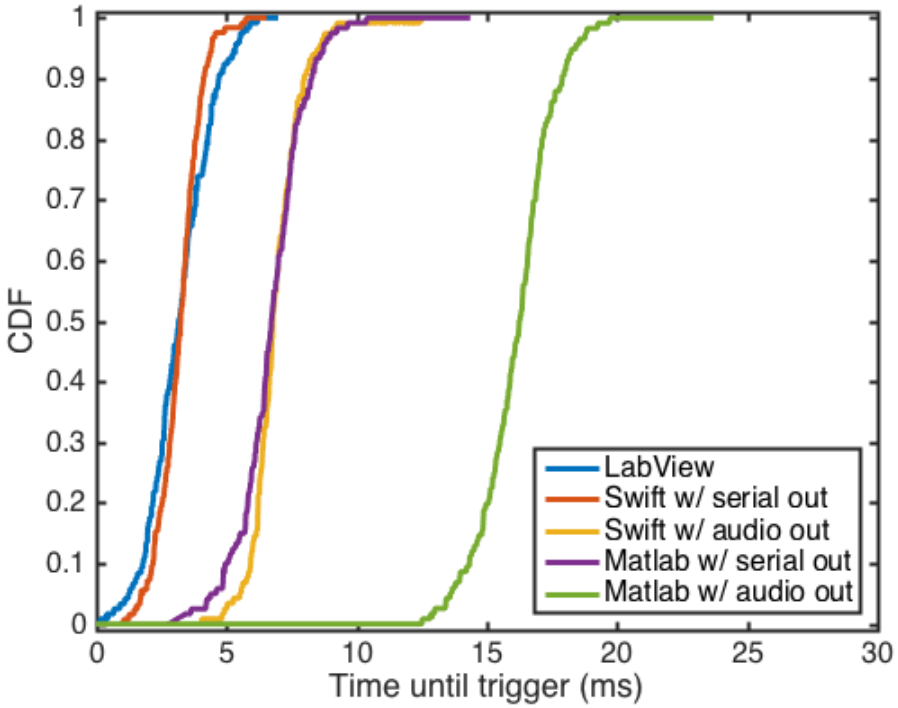

Figure 4. Timing curve for a syllable running a test file generated by training.

frequency—at very low computational cost. However, we have neither described this in detail nor characterised its performance.

Duration detection is even easier. The beginning and end of a syllable may be detected by looking at the ratio of total energy in the singing frequency band (approximately 1.5-7 kHz) to the total energy, over some small time window. Any syllable thus identified that also contains a trigger event may be monitored for duration.

In feedback experiments such as frequency- or duration-shifting, vocal output changes over the course of the experiment. The neural network excels at identifying syllables close to its training set, so as vocal output changes the detector may not recognise a match. If the detector must be robust to this shift, it may be retrained every so often as the bird learns, or data consisting of synthetically pitch-shifted or duration-shifted target syllables over the recognition region may be added to the training set. We will test these approaches in future work.

6 Acknowledgments

But by whom?

This work was funded by someone.

A Resources

PLOS doesn't really say where such things should go.

Cleanup: mostly done. But it would be nice to move all this to the lab's github group.

Song alignment: Last we checked, Jeff Markowitz's song alignment software could be found at `https://github.com/jmarkow`.

Training the neural network: Our implementation of the syllable detector training code is available under the GNU GPL at: `https://github.com/bwpearre/`

Runtime: The Matlab, Swift, and Labview implementations for executing the trained network: `https://github.com/nathanntg/syllable-detector`

References

1. Leonardo A, Konishi M. Decrystallization of adult birdsong by perturbation of auditory feedback. Nature. 1999;.

2. Andalman AS, Fee MS. A basal ganglia-forebrain circuit in the songbird biases motor output to avoid vocal errors. Proceedings of the National Academy of Sciences of the United States of America. 2009;Available from: `http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2709669/`.

3. Keller GB, Hahnloser RHR. Neural processing of auditory feedback during vocal practice in a songbird. Nature. 2009;.

4. Skocik M, Kozhevnikov A. Real-time system for studies of the effects of acoustic feedback on animal vocalizations. Frontiers in Neural Circuits. 2013 January;6.

5. Poole B, Markowitz JE, Gardner TJ. The Song Must Go On: Resilience of the Songbird Vocal Motor Pathway. PLOS One. 2012 June;7(6). Available from: `http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0038173`.