

Recognizing Phonemes in Continuous Speech - CS640 Project

Kate Ericson

May 14, 2009

Abstract

As infants, we hear continuous sound. It is only through trial and error that we eventually learn phonemes, recognize them, and then start trying to string them together. I attempt to train a Liquid State Machine (LSM) to achieve the second step in the process – picking out particular phonemes from continuous speech. I take several different approaches to explore how various settings within the LSM change the learning process.

Contents

1	Introduction	1
2	Experimental Setup	2
2.1	LSM design	2
2.2	Auditory Encoding	3
2.3	Spike Coding	4
2.4	Tests	5
3	Results	7
4	Conclusions	8
5	Future Work	8
6	References	9

1 Introduction

While the task of phoneme recognition can be considered as a simple classification problem, it is actually a more involved task. There is an inherent temporal nature of the problem that is not easily accounted for with most artificial learning approaches. A Liquid State Machine (LSM) is uniquely suited for this task,

as it contains a pool of neurons that allows it to effectively have a short-term memory [3].

Infants originally hear only continuous speech – it is only through trial and error that an infant eventually learns how to break this apart into phonemes, and eventually from there learn words [1]. Computers are effectively at this same stage – all sounds are continuous. By providing a more biologically inspired approach to speech recognition (via LSMs), it may be possible to further advance the field of Natural Language Processing (NLP).

In this particular experiment, a LSM was trained to recognize the /@/ phoneme in one set of runs, and the /dh/ phoneme in another set. The MOCHA-TIMIT speech corpus [9] was used for all tests. The Lyon Passive Ear Model, a biologically inspired model of the inner-ear [7], was used to process the input speech, and input as an analog signal into the LSM with Leaky Integrate and Fire (LIF) neurons.

The organization of the rest of the paper is as follows: In section 2 I will discuss the experimental setup, and will also discuss the various LSM designs I use, as well as the steps I take to preprocess the sound files. In the following section I present the results of my experiments, and discuss what they mean. I then conclude with my ideas for future work.

2 Experimental Setup

The experiments were performed using CSIM [6], a neural simulator written in Matlab, as well as Auditory Toolbox [7], a toolbox of auditory models also written in Matlab. As inputs, the MOCHA-TIMIT speech corpus [9] was used. I am using both speakers that MOCHA-TIMIT supplies: the female with a southern English accent and male with a northern English accent recordings.

2.1 LSM design

For the Liquid State Machine, I used two different configurations. Both are similar to the one described in [4]. For my main pool, I used a pool of randomly connected Leaky Integrate & Fire (LIF) neurons located on the integer points of a 15 X 3 X 3 column. As I am training for the recognition of only one phoneme, I have only one output neuron. In my first configuration, it is attached to the last 3 X 3 layer of the main pool. In the second configuration, the output neuron was attached to all 135 neurons in the main pool.

For the second configuration, my connection weights and probabilities match those discussed in [8]. As my task of phoneme recognition is similar to their task of word recognition, it made sense to use a similar setup.

Through testing, I found that these settings were too sparse for my first configuration. The probability of two neurons a , and b having a synaptic connection between them is given in this equation:

$$P_{conn}(a, b) = C * e^{\frac{-D^2(a,b)}{\lambda^2}}$$

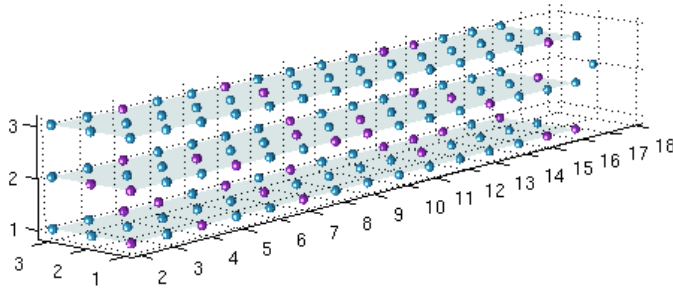


Figure 1: Basic configuration - does not include input neurons, and connections are not shown

where D is the euclidean distance between neurons a and b , and C is a constant that varies depending on whether a and b are excitatory (E) or inhibitory (I): 0.3 (EE), 0.2 (EI), 0.4 (IE), 0.1 (II). λ controls the average number of connections, as well as the average length of connections between neurons. Using the settings defined in [8], I originally tried the λ setting of 2. This, however, allowed for “dead spots” in my main pool, and signals were lost before reaching the last part of the pool that my output neuron was connected to. To fix this problem, I set $\lambda = 4$. This not only increased the number of connections in my main pool, but also increased the average length of connections – allowing signals to pass all the way through the pool.

The biggest difference between my setup and the two papers mentioned above is how I handled input. The experiments with isolated word recognition generally performed an extra processing step, where analog signals were turned into spike trains, and only the 40 spike trains deemed most useful were actually used. I simply ran my sound files through a Lyon Passive Ear Model implementation found in [7], and then used the resulting cochleagram as analog inputs to my LSM.

2.2 Auditory Encoding

While the standard technique for preprocessing speech uses Mel-Frequency Cepstral Coefficients (MFCC), I decided to use the Lyon Passive Ear model. The

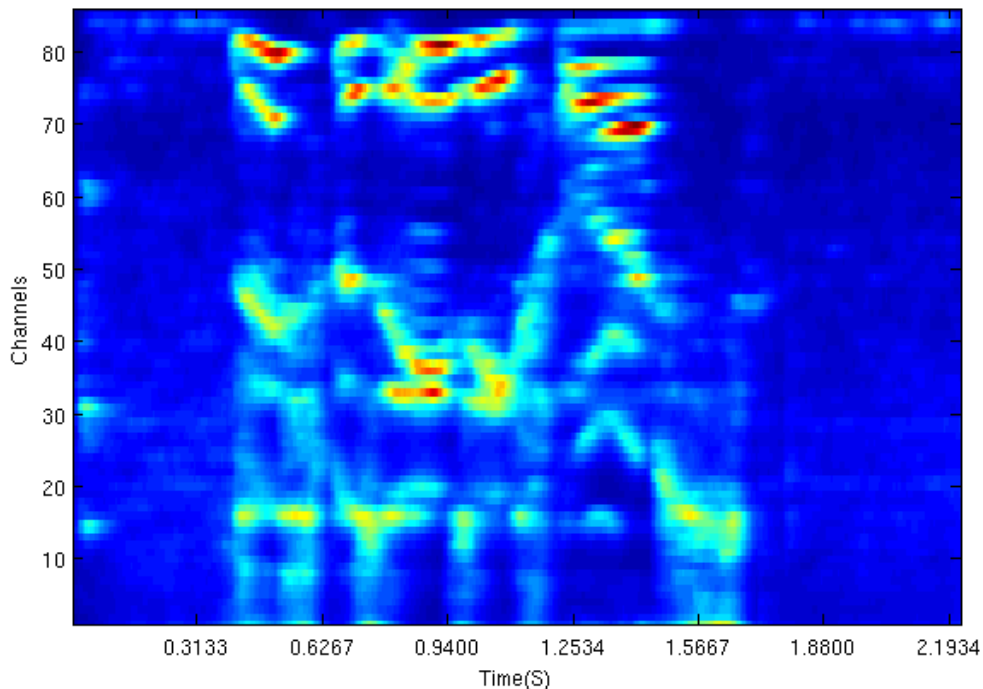


Figure 2: The cochleagram resulting from: "This was easy for us." Output of the Lyon Passive Ear model implemented in [7].

Lyon Passive Ear [2] is a biologically inspired model of the inner ear. This model calculates the probability of firing along the auditory nerve as a result of an input sound at a given sample rate [7]. An example cochleagram generated with the sentence "This was easy for us." can be seen in 2. The cochleagram returns an array that displays spiking probabilities for 86 different channels across time. While this form of processing is computationally more expensive than other more traditional methods of speech processing such as MFCC, it has been shown to be more robust in the presence of noise [8].

2.3 Spike Coding

Generally, when a preprocessing algorithm returns an analog signal a spike encoding algorithm is used to translate an analog signal into a spiking signal before it is used as input to a recognition system. Two of the more commonly used tools are BSA (Ben's Spiking Algorithm) and Poisson spike trains. These encoding algorithms step through an analog input, and decide whether a spiking signal should be sent based on the data. CSIM [6] has a special Analog Input

Leaky Integrate and Fire neuron that performs similar computations on analog input.

At this time, all spike coding is handled by a pool of Analog Input Leaky Integrate and Fire neurons. In my first configuration, these are densely connected to the first 3 X 3 layer of my main pool. In my second configuration, I have all input neurons connected to all neurons in my main pool. Again, this is more computationally expensive than other methods (such as BSA or Poisson spike trains) [8], but it is also a more biologically realistic model. I use the 86 analog signals returned by the Lyon cochleagram as direct inputs into my analog input pool.

The number of channels returned (86 in my case) is determined by the Auditory Tool Box [7]. The more channels there are, the more accurate the model becomes. This number is bounded by the quality of the input. All MOCHA-TIMIT [?] sound files have a frequency of 16kHz, so the model produces 86 samples.

2.4 Tests

Both LSM configurations were trained to recognize */dh/* and then */@/*. For both configurations, training sets of 400 inputs and test sets of 60 were used. In these experiments, I tested on only one speaker (each speaker in MOCHA-TIMIT has 460 sentences). The fully connected (second) configuration could not handle larger amounts of training data, so I only used my sparsely connected (first) configuration for training/testing across speakers.

To better view the problem, the cochleagrams of "This was easy for us." with the */dh/* and */@/* phonemes mapped onto it can be seen in Figures 2.4 and 4

With the sparsely connected configurations I trained on 800 inputs, and tested on the remaining 120 sentences. The training set contained all sentences from one speaker, and 340 sentences from the other. Since both of the speakers in MOCHA-TIMIT go through the same sentences, the LSM had seen the 120 test sentences before, but from the other speaker. In both cases, I still trained the LSM to recognize the frequently occurring */@/* and the rarely occurring */dh/*.

In all cases, I used a Linear Classification model that is part of the CSIM [6] package for training. This model is designed to run in batch training mode, and after completing its training and test sets, returns the average mean squared error (MSE) for training and test sets. This is scaled to 1, with 1 being the worst – the pattern was never matched, and 0 being the best – the pattern was perfectly matched. During training, only the weights connected to the output neuron are modified – all weights internal to pools are left alone.

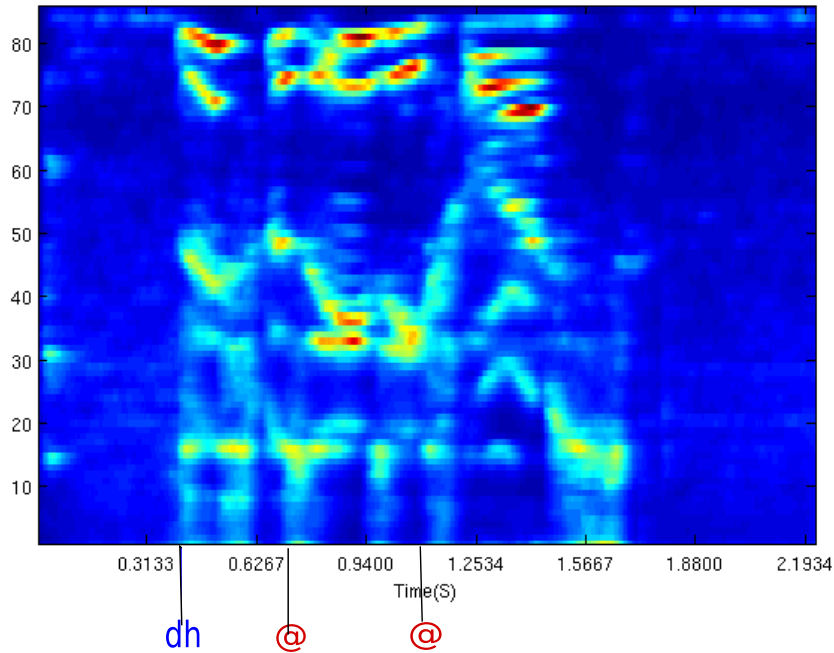


Figure 3: Female cochleagram of "This was easy for us." with /*dh*/ and /*@*/ phoneme onset location shown

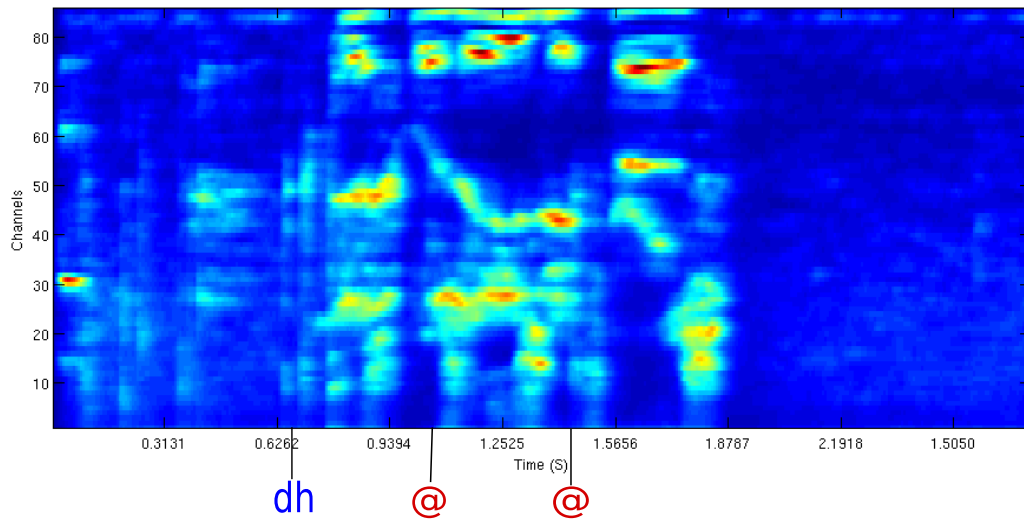


Figure 4: Male cochleagram of "This was easy for us." with /*dh*/ and /*@*/ phoneme onset location shown

	/dh/		/@/	
	male	female	male	female
fully connected	0.491275	0.494634	0.491275	0.494634
sparsely connected	0.491275	0.494634	0.491275	0.494634

Table 1: Test 1: MSE reported from test runs for both male and female speakers from MOCHA-TIMIT on /dh/ and /@/

	/dh/	/@/
testing on 120 sentences from female	0.49249	0.492958
testing on 120 sentences from male	0.453309	0.496084

Table 2: Test 2: MSE reported from sparsely connected LSM on /dh/ and /@/. 800 training sentences, 120 test sentences

3 Results

As LSMs have many stochastic qualities, I performed several runs for each test, and then calculated the average. In each case, Mean Square Error (MSE) from the test set is reported. A MSE of 1 means the correct output was never achieved, while a MSE of 0 means that the correct output was met exactly. A score of .5 is about what we can expect from randomness.

With the smaller test sets 3, the model performs just barely above random, and has the same MSE for detecting either phoneme regardless of LSM configuration

In the larger tests 3, the scores show a slight improvement over the smaller tests, but still are not very far from random. It also seems to perform better across the board on the male data set than the female data set.

A sparsely connected configuration can be trained far more quickly than a fully connected LSM. The average time to go through 50 simulations for the sparse configuration is about 300 seconds, while it takes an average of about 1500 seconds for the fully connected LSM. A sparse configuration can also handle a larger training set without eating up too much memory. Should I move to an online training algorithm, however, both points may become moot.

With the smaller training set that both configurations could handle, the fully connected LSM was better able to focus on specific spiking times, and capture phonemes that do not occur often. The sparsely connected configuration would never spike for /dh/, only for /@/.

Neither configuration was able to clearly distinguish between the phoneme it was looking for and non-phoneme as I would have hoped. Instead of being able to tell precise spiking times from the output, I was only able to tell that whether or not the phoneme occurred in the sentence.

4 Conclusions

Through this study I learned several things. As far as LSM configurations are concerned, I found that sparsely connected LSM configuration is best for quick implementations, and only starts to show meaningful results after a large amount of training runs. A more fully connected LSM, on the other hand, requires less training runs to start returning meaningful results and is also capable of picking up on less frequently occurring patterns.

In [5], a generic LSM configuration is proposed based off of unpublished lab data. In my experiments, I found that this generic LSM configuration with a main pool of 135 LIF neurons arranged in a 15x3x3 pile is not capable of recognizing phonemes with enough clarity to be particularly useful. In the future, I hope to approach this problem again with a different main pool configuration to see if I can gain that clarity.

The LSM seemed to perform better with the female data set after being trained on the male data set than the other way around. I am not too sure what to make of this. There are several possible reasons that this is happening. First, the LSM may have an easier time finding phonemes after training on lower frequency sounds than the other way around. Another reason might be because of the accent difference. The female and male speakers have slightly different accents – it might be easier for the LSM to switch from the male speaker’s accent to the female speaker’s accent.

5 Future Work

An initial future goal is to add other output neurons and train a network to recognize multiple phonemes. I want to be able to completely map sentences into phonemes. In the long run, I feel that this will provide the most contributions to the field of NLP. I would also like to work with a larger speech corpus, possibly extending to different accents.

For the sake of making my model as biologically realistic as possible, I chose many computationally expensive algorithms. I also used batch training algorithms, which also added to the amount of time needed to obtain a working model. In the future, I would like to not only implement an online training algorithm, but also see about parallelizing some of the processing that I need to run. In an ideal situation, I would be able to speed up the process to the point where I can get real-time feedback on speech as it is being said.

In the future, I would like to retry this experiment with a main pool of a different size. As mentioned in [3], the 15x3x3 pool that I am currently using is a generic size. There’s a chance that I would be able to find either smaller configurations that work as well as what I currently have, thus cutting down on system resources necessary to run, or possibly larger configurations that can more accurately report the beginning and end of phonemes.

Another thing I would like to look at is the difference between male and female speakers. I noticed that when my sparse LSM was trained with the male

speaker and partially with the female speaker it performed better than when it trained with the female speaker and part of the male speaker. I would like to isolate this occurrence and figure out why it is happening – I want to know if it’s because of the male/female vocal range differences, because of the accent differences, or something else entirely.

6 References

References

- [1] IEEE/RSJ International Conference on Intelligent Robots and Systems, *Segmenting acoustic signal with articulatory movement using recurrent neural network for phoneme acquisition*, Nice, France, Sept 2008.
- [2] R. F. Lyon, *A computational model of filtering, detection, and compression in the cochlea*, Proceedings of IEEE-ICASSP-82, 1982, pp. 1282–1285.
- [3] W. Maas, T. Natschläger, and H. Markram, *Real-time computing without stable states: A new framework for neural computation based on perturbations*, Neural Computing (2004), no. 14, 2531–2560.
- [4] W. Maass, T. Natschläger, and H. Markram, *A model for real-time computation in generic neural microcircuits*, Proc. of NIPS 2002, vol. 15, 2003, pp. 229–236.
- [5] T. Natschläger, H. Markram, and W. Maass, *Computer models and analysis tools for neural microcircuits*, Neuroscience Databases: A Practical Guide,, Kluwer Academic Publishers, 2003, pp. 123–138.
- [6] Thomas Natschläger, *Csim: A neural circuit simulator*, 07 2006, Matlab tool.
- [7] Malcolm Slaney, *Auditory toolbox*, 1998, Version 2, Interval Research Corporation.
- [8] D. Verstraeten, B. Schrauwen, D. Stroobandt, and J. Van Campenhout, *Isolated word recognition with the liquid state machine: a case study*, Information Processing Letters **95** (2005), no. 6, 521 – 528, Applications of Spiking Neural Networks.
- [9] Alan Wrench, *Mocha-timit*, November 1999, <http://www.cstr.ed.ac.uk/research/projects/artic/mocha.html>.