An Electrophysiological Neuronal Cell Classifier

Lane McIntosh

June 7, 2010

1 Project Background

In general, there are three primary ways one can classify a cortical interneuron - either by its morphology, chemistry, or electrophysiology. Towards the goal of elucidating microcircuits in the cortex - small networks of neurons working together to synthesize information and produce singular output - the most relevant and interesting classification is the identification of functional neuronal cell types. The use of "functional" in this context indicates the differentiation of cells based on evidence of their electrical behavior; this is essential because it is this activity - specifically, the neuron's output pattern of action potentials - that affects the behavior of neighboring neurons and the operation of the cortical circuit.

There is also the question of generalizability of classification - is there any overlap between electrophysiological (functional) classification and morphological or molecular/chemical classification? The morphological differentiation of interneuron classes has been well established for instance - there are Chandelier cells, large basket, nest basket, and small basket cells, double bouquet cells, bipolar cells, bitufted cells, and Martinotti cells. In a 2004 review by Markram et al., electrophysiological distinctions did not necessarily represent morphological or chemical categories, but nonetheless the review found a great deal of overlap between these morphological categories and chemical and electrophysiological types (see Figure 1). [1]

In terms of this electrophysiological classification, there are six general categories according to the Petilla nomenclature (2008). These categories are based both on spike rate and on the regularity of spike temporal patterning; the categories are 1) Fast spiking cells, 2) Non-adapting, non-fast spiking, 3) Adapting, 4), Irregular spiking, 5) Intrinsic burst firing, and 6) Accelerating spiking cells. In addition to these six categories, the Petilla nomenclature sorted spike patterns into four additional sub-categories based on whether the patterns were bursting, continuous, delayed, or stuttering (see Figure 2). [2]

Different authors have formed electrophysiological classes and subclasses differently. In the Markram et al. review (2004), five classes are formed based on the steady-state responses of bursting (BST), accommodating (AC), non-accommodating (NAC), stuttering (STUT), or irregular spiking (IS). The authors then created burst (b), classical (c), initial (i), repetitive (r), and transient (t) subclasses based on onset response types. In addition to this classification scheme, other classifications also include late spiking (LS), burst spiking non-pyramidal (BSNP), and regular spiking non-pyramidal (RSNP) categories.

However, this treatment of a neuron's electrophysiological output is simplified and theoretical. In real cells, there is often variation in the amplitude and half-width of an action potential, there are often dendritic potentials following spike patterns, and there are even afterhyperpolarizations (AHP amplitudes) that can also be used to classify cells. Additional parameters that are useful for classifying interneurons include input

[1] Markram, Henry, Maria Toledo-Rodriguez, Yun Wang, Anirudh Gupta, Gilad Silberberg, and Caizhi Wu. "Interneurons of the neocortical inhibitory system.." Nature reviews. Neuroscience 5, no. 10 (2004): 793-807. doi:10.1038/nrn1519. http://www.ncbi.nlm.nih.gov/pubmed/15378039.

[2] Ascoli, Giorgio A, et al. "Petilla terminology: nomenclature of features of GABAergic interneurons of the cerebral cortex.." Nature reviews. Neuroscience 9, no. 7 (2008): 557-68. doi:10.1038/nrn2402. http://www.ncbi.nlm.nih.gov/pubmed/18568015.

Figure 1: Overview of Classification Types - Morphological, Chemical, and Electrophysiological.

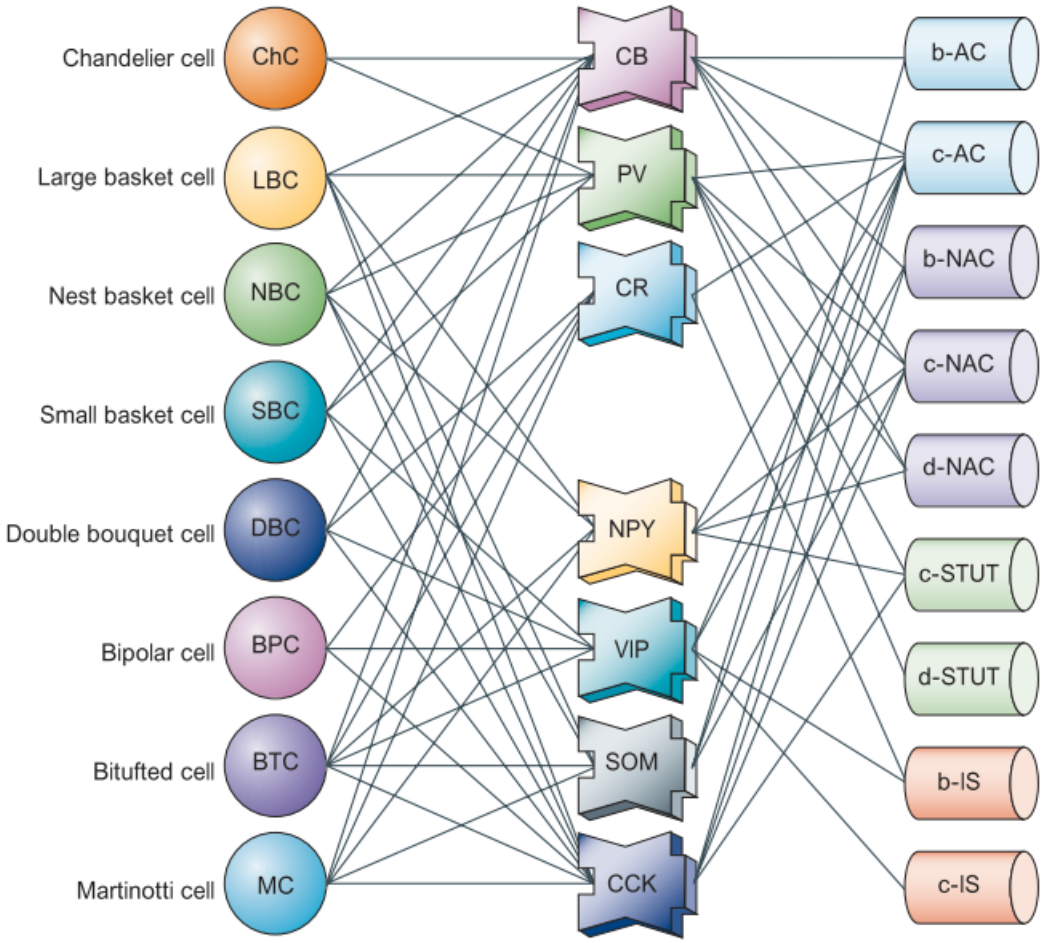
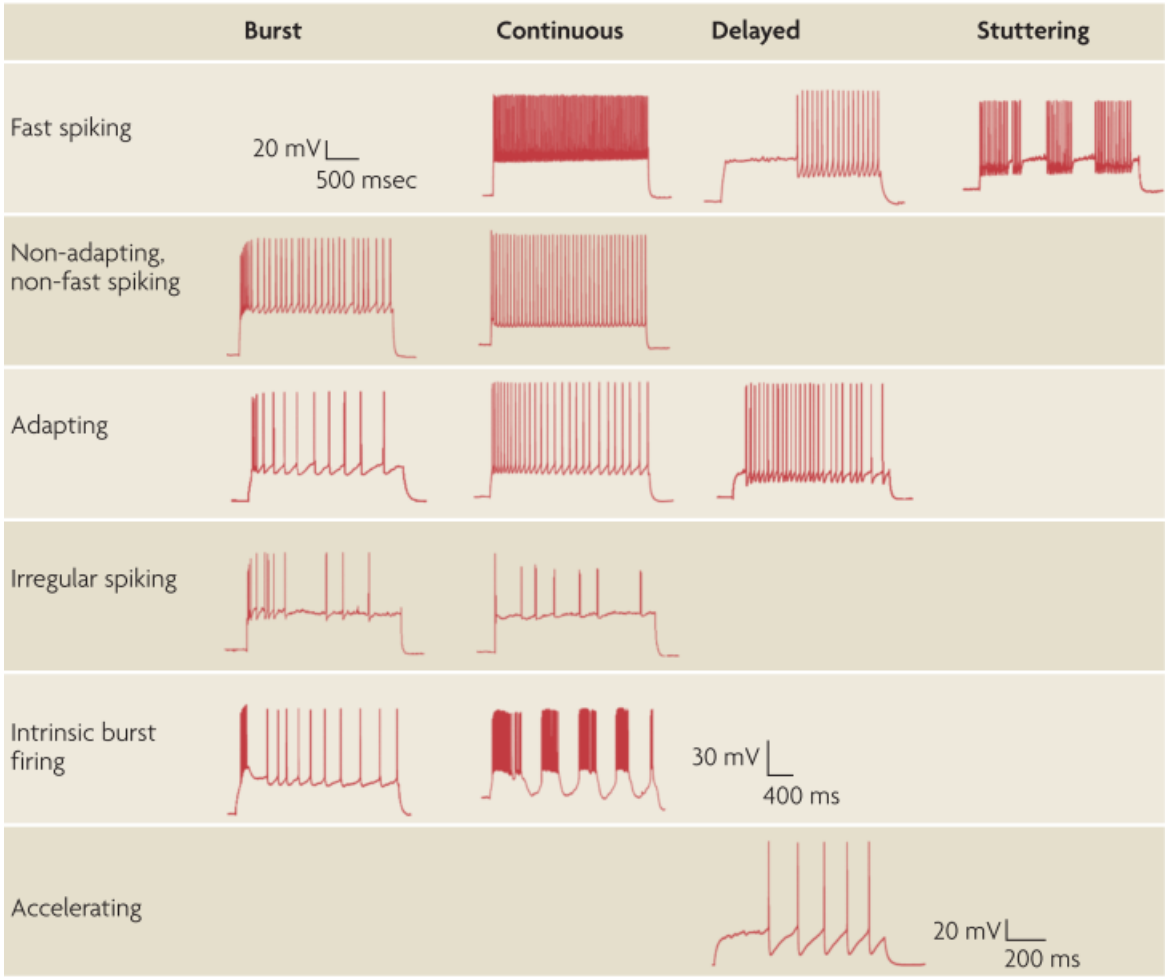


Figure 2: Petilla Terminology - Electrophysiological classification based on spiking pattern.



resistance R_N, the threshold at which spiking can occur, and the number and degree of spike-frequency accommodations (SFAs). [3]

2 Proposed Parameters

Electrophysiological classification is based on the cell's response to depolarizing current steps; since we are injecting current and measuring the cell's voltage output, we perform this in current clamp. The only parameter mentioned so far that is not measured from this type of data is the input resistance R_N which is collected from voltage clamp recordings of the interneuron's passive properties. In addition to measuring input resistance, it might be interesting to see in voltage clamp whether or not the cell is Ohmic (has a linear I-V relationship), and whether or not this correlates with any classification paradigms.

In order to involve both idealized spike patterns (spike rate and temporal patterning) and deviations from the theoretical model of spiking (where the interneuron's output is something other than a train of Dirac delta functions) into our classification scheme, we will use the following parameters (see table below):

Type of Data Collection	Parameter	Considers spike as idealized delta function?
Voltage clamp	Input resistance (R_N)	No
Voltage clamp	Capacitance	No
Voltage clamp	I-V relationship (linear?)	No
Current clamp	Action potential firing rate	Yes
Current clamp	Distribution of inter-spike intervals (ISIs)	Yes
Current clamp	Spiking threshold	No
Current clamp	Peak action potential amplitude	No
Current clamp	Action potential width at half-peak	No
Current clamp	Latency of response	Yes or No
Current clamp	AHP amplitude	No

3 Objectives

One caveat to this proposal is that the data collected consists solely of electrophysiological data (and so there is no other way of comparing our classifications to previous literature), and the electrophysiological data consists of perturbing the cell with 500ms or 1sec duration current steps - and so it is questionable whether or not it is possible to extrapolate the properties inherent in a spike train (e.g., if it is bursting, irregular spiking, accommodating, accelerating, etc.) from such a short depolarizing current.

Given this consideration, the primary objective will be to see if these different parameters mentioned above vary across a continuum or if they do exhibit clustering (and are subject to non-arbitrary classification), and if we can reduce the dimensionality of our parameters (for instance, using PCA). This latter question would be useful if we wanted to see the interaction of the various electrophysiological properties and what parameters might be redundant for classifying cells.

[3] 1. Raymond A. Chitwood and David B. Jaffe, Calcium-Dependent Spike-Frequency Accommodation in Hippocampal CA3 Nonpyramidal Neurons, J Neurophysiol 80, no. 2 (August 1, 1998): 983-988.

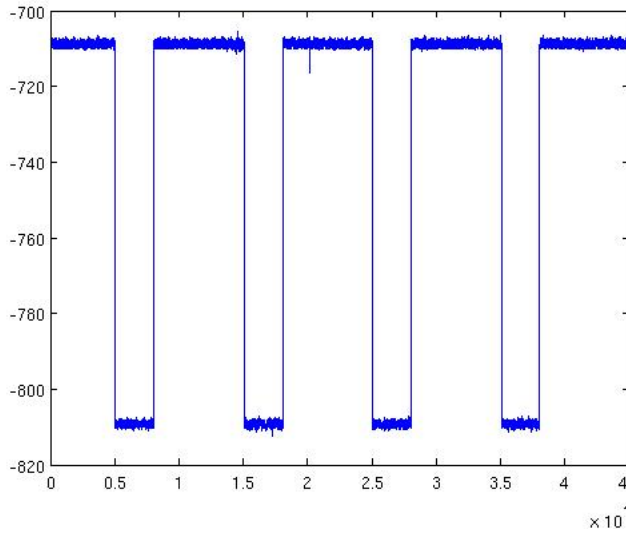


Figure 3: Voltage trace: Passive Properties

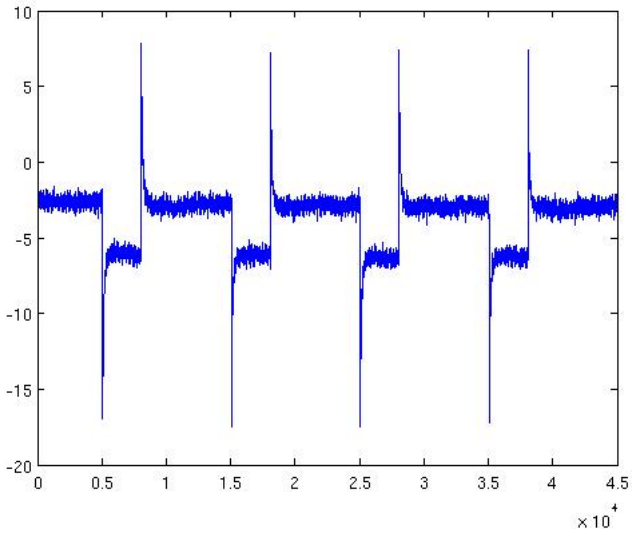


Figure 4: Current Trace: Passive Properties

4 Data

All the initial data that the classification algorithms will be based on data courtesy of Peter Kruskal. A sample of the data is provided below.

Figures 3 and 4 are selected voltage and current traces under voltage clamp, illustrating the passive properties of the interneuron in question.

Figures 5 and 6 are also voltage and current traces, but under current clamp; these are "spike tests."

5 Preprocessing

The very first step in data preprocessing was to automate selecting spike test voltage and current traces from the different available channels. Because the spiking and current steps seen in the spike tests has large variance, this was used to identify whether a voltage or current trace was from a spike test.

The next step towards executing the project objectives was to deconstruct the data into useful parts and helpful simplifications. As shown in figures 5 and 6, the original data is in the format of raw voltage and current traces sampled at 10,000 Hz. Since these are directly measuring the voltage output and current input of the experimental preparation (the patch clamped neuron), these traces have both technical and biological noise that do not contribute to the neuron's reproducible functionality. In addition, we will want to split our analysis along two roads - the first where we analyze only an idealized train of spike times from the neuron (ignoring all other voltage behavior, spike amplitude and width, etc.), and the second where we analyze the biological deviations from this idealized behavior (variation in spike amplitude and width, latencies, etc.). Since we just have a series of data points (with one every 1/10000 second), we first wrote an algorithm that detects spiking and converts it into a train of spike times.

A raster plot of the spike trains from all 39 neurons is shown in Figure 7. In this figure, each spike is

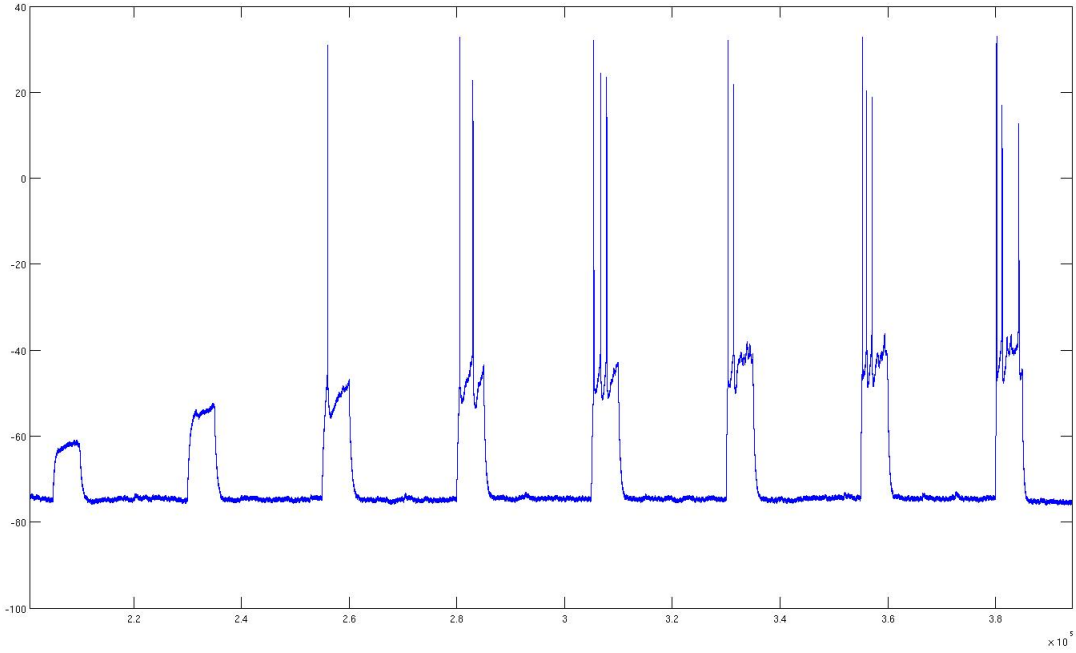


Figure 5: Voltage trace: Spike Test

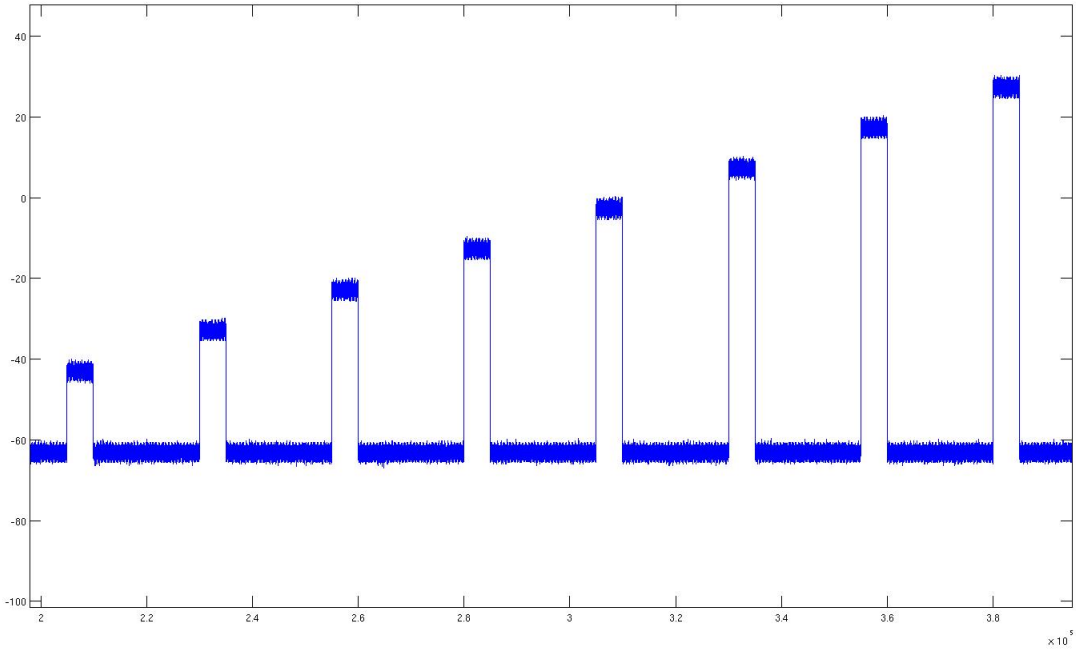
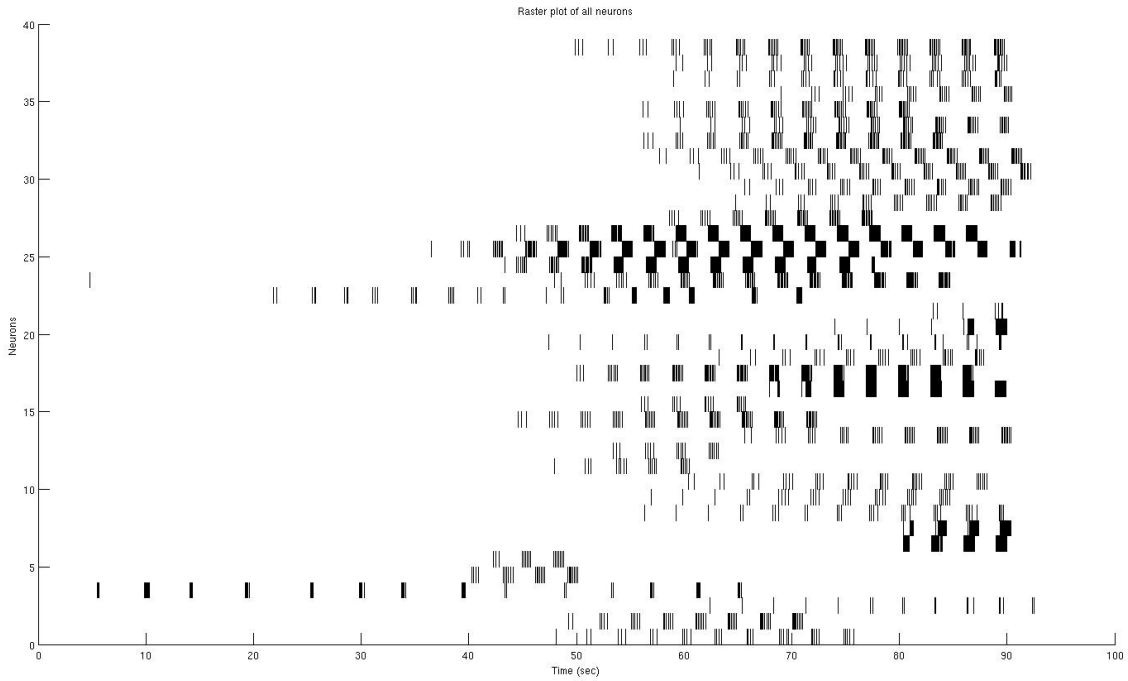


Figure 6: Current Trace: Spike Test

Figure 7: Raster Plot of Spike Trains for all 39 Neurons.



represented by a single line 1 unit in height. The x-axis represents the real time in seconds of the neuronal recording from 0 to 100 seconds (although each neuron is recorded for a different length of time ranging from 70 to 100 seconds). Each horizontal train of rasters is from a single neuron.

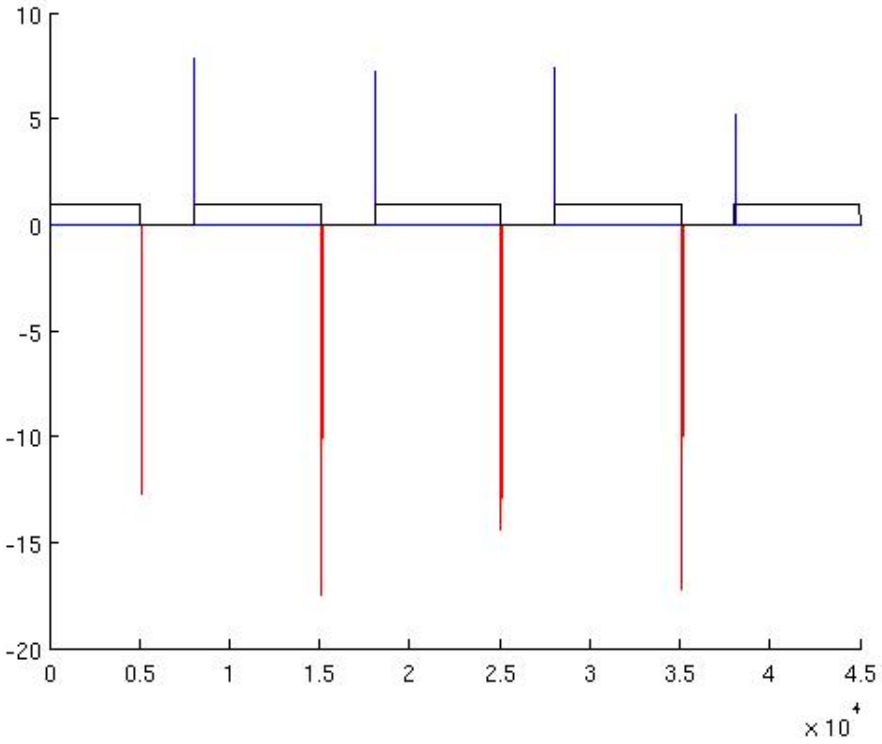
This spike-detection algorithm first employs a data smoothing step whereby sample-to-sample variability is reduced and where single and double time point reversals in voltage direction are removed (to give a comparison of scale, a typical spike is on the order of 100 time points in half-width).

The next step was to visualize the spikes in a biologically relevant context; specifically, to find the spikes as a function of the injected current steps.

Performing a more extreme data smoothing on the current steps (since we only care about the current step amplitude, duration, and the exact time when it started and ended regardless of how we are analyzing the voltage trace). This was performed by having a train of 0's and 1's corresponding to when there was a current step, then multiplying this train to the actual current trace, then smoothing each step to its mean current. After this was accomplished, I constructed a cell array of "go" times - i.e., a list of times when each current step first started. Since the baseline current was different in every cell, according to the input resistance of a given neuron, the current was normalized to a baseline of 0 pA at the beginning.

In addition to these start times, I also paired each start time with the normalized current step amplitude, the vector of 0's and 1's, and the unnormalized current step amplitude (for possible use later in determining input resistance).

Figure 8: Component Preprocessing of Voltage Trace.



Turning again to the voltage trace, I performed a parallel data preprocessing step for use in the non-idealized voltage analysis. There are three crucial components that it is necessary to distinguish - let's call them points a, b, and c. Point a corresponds to the steady state reached just before the spike, point b corresponds to the peak of the action potential, and c corresponds to the steady state reached after the action potential, just before the current steps back down. With these three points we can navigate through the voltage trace and find what we need to. For instance, to calculate the decay of an action potential, we can specify that we'll model an exponential decay to the window from b_i to c_i. To calculate the input resistance, we could find the time constant τ associated with the double exponential between c_i and a_i. These three components are shown in Figure 8 - the spike is in blue, the action potential decay window before current steps down is in black, and the instance immediately after point c_i is in red.

In Figure 8, along the x-axis are the data points (1/1000 sec), and the y-axis is normalized voltage (normalized just for the purpose of demonstrating the three component parts).

6 Implemented Parameters

The proposed parameters suggested above in section 2 were theorized to capture the variance of a given neuron's voltage recordings. As these parameters were being extracted from the data, certain qualitative

patterns emerged (for instance, realization that action potential amplitudes after the first spike in a given current step roughly follows a logarithmic curve), that enabled us to quantify the extracted data structure with a variety of metrics related to the proposed parameters.

Accordingly, there were three levels to my approach towards variation within distributions of the above hypothetical parameters. The first and simplest is averaging, which does not capture the inherent structure of the variation; the second is finding the standard deviation of the parameter distribution, which provides a measure of the variability across the parameter as well as a crude measure of adaptation (in the case of firing rate) or dendritic bursting (in the case of action potential widths). The last level was to fit a curve to the parameter distribution if there was some theoretical curve it should fit, or else if qualitative patterns emerged from the parameter distribution that could be modeled. The most often used curves were linear fits and logarithmic fits.

Also, as a final note to this section, we were able to calculate input resistance (and the V-I relationship) from only current clamp data by using intermediate negative current steps and comparing these normalized current steps with each respective average (passive) voltage displacement step. Then taking the ratio of $\Delta V/I$ we find input resistance under Ohm's Law.

All of the implemented parameters in the below table were extracted from the spike test data in current clamp. The implementation of the proposed parameters resulted in 11 variables (or metrics) per neuron (and so dimension = 11).

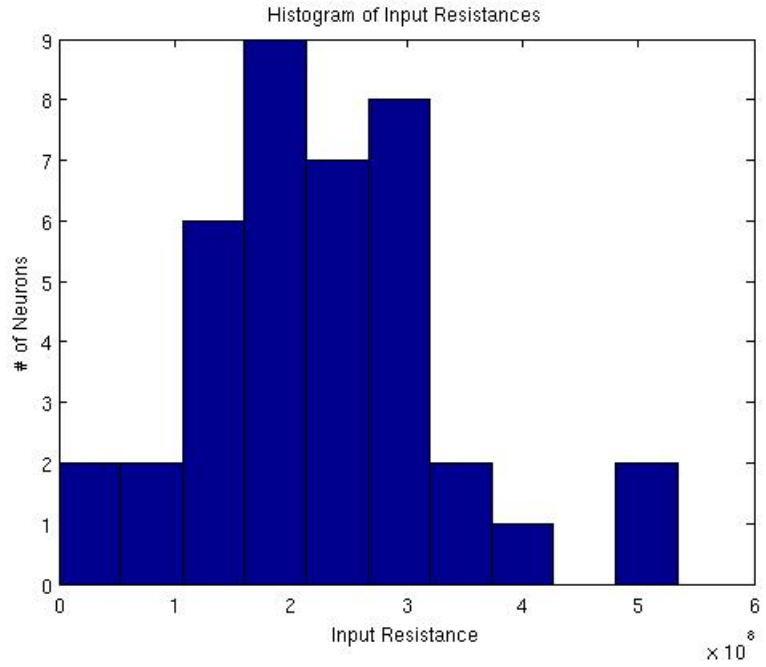
Implemented Metrics	Level of Analysis
Input resistance (R_N)	Mean R_N across 3 small negative current steps
V-I relationship linearity	Linear curve fit: how linear is the relationship?
Action potential amplitude logarithmicity	Log curve fit: how well do subsequent spike amplitudes fit the curve?
Action potential amplitude standard deviation	Average standard deviation of spike amplitudes within current step
Action potential width	Average time (sec) duration of spikes at half-width
Action potential width standard deviation	Average standard deviation of within-current step spike widths
Spiking threshold	Spiking threshold in voltage (from R_N and min. active current step)
Firing rate	Average firing rate (across all active current steps; 1/ISIs)
Firing rate standard deviation	Average standard deviation of firing rates
Latency of Response	Average latency (time until first spike after current steps on)
Latency logarithmicity	Log curve fit: how logarithmic are the latencies across a current step

7 Metrics and Analysis

Let's first look at the metrics associated with the original data before it is simplified to an idealized spike train; we will examine each metric from the table above consecutively. The first of these metrics is input resistance (Figure 9). Input resistance is on the scale of hundreds of mega-ohms, and the distribution looks Gaussian, with two exceptionally large input resistances. Since the larger the voltage displacement the larger the active properties, I took input resistances from the 3 smallest negative current steps.

If we take the input resistance across all passive current steps (i.e., the current steps where there is no corresponding spiking), we can construct a plot of voltage difference as a function of current. This is a V-I plot, and the slope corresponds to the input resistance. Ohmic relationships are represented by linear V-I plots. We can take a linear fit and calculate the least squares error between the data and the linear regression. Taking this error parameter, where a larger value corresponds to a less linear V-I relationship, we can plot

Figure 9: Histogram of Input Resistances



the histogram across all neurons (Figure 10). Interestingly, we get a similar distribution as in Figure 9, with a fairly normal distribution and one (or two) large outliers.

To get a feel for the typical linearity of a V-I plot, V-I curves are plotted for two sample neurons in Figures 11 and 12.

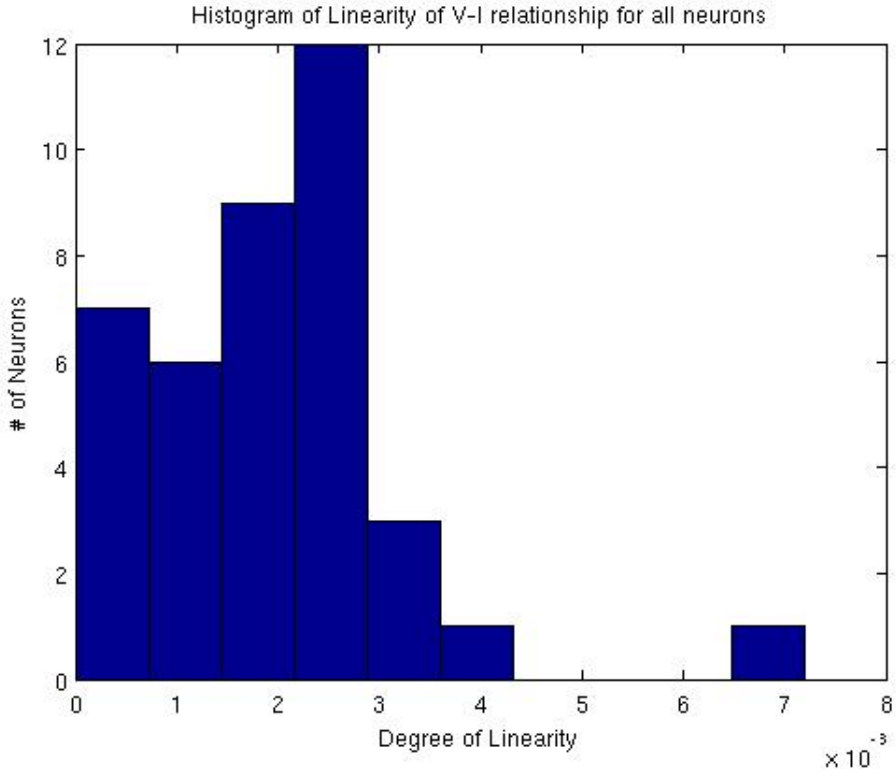
Next I determined the voltage amplitudes for each action potential and looked at the distribution of action potential amplitudes for each neuron. The distribution of amplitudes for one sample neuron is shown if Figure 13.

In Figure 13 we have the action potential amplitude in mV on the y-axis and the spike number (chronologically ordered) on the x-axis.

Looking at the graph qualitatively, we can see that for each current step, the first spike has a large amplitude, and then subsequent spikes during the same current step have smaller amplitudes. This pattern seemed fairly uniform across the neurons, and moreover the distribution is highly irregular, and so thinking of a way to model the distribution that captures the variability between different neurons has proved difficult. For the moment I'm sticking with looking at the mean and standard deviation of the amplitudes from the first spikes, and the mean and standard deviation of the amplitudes from the subsequent spikes, separately.

Accordingly, we can fit the amplitudes of action potentials after the first spike amplitude in each current step to a logarithmic curve and find the least squared error. This can be thought of as the "logarithmicity" of the distribution of action potential amplitudes, where a larger error value indicates a poorer fit to the

Figure 10: Histogram of V-I Linearity



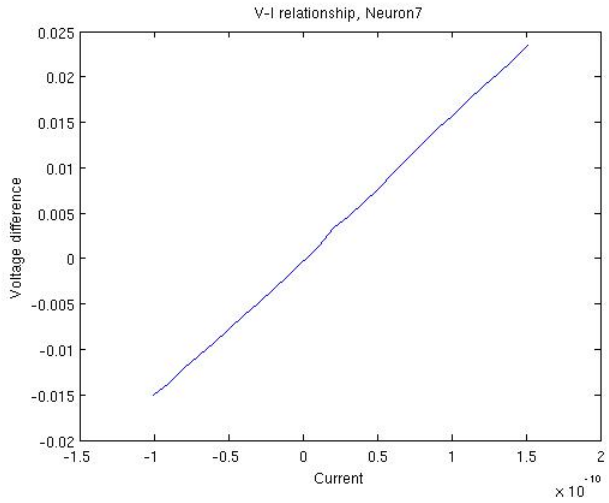


Figure 11: V-I Curve: Neuron 7

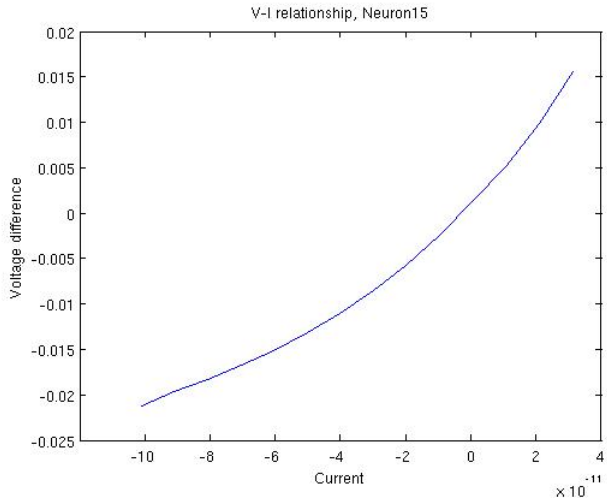


Figure 12: V-I Curve: Neuron 15

Figure 13: Distribution of Action Potential Amplitudes for a Sample Neuron.

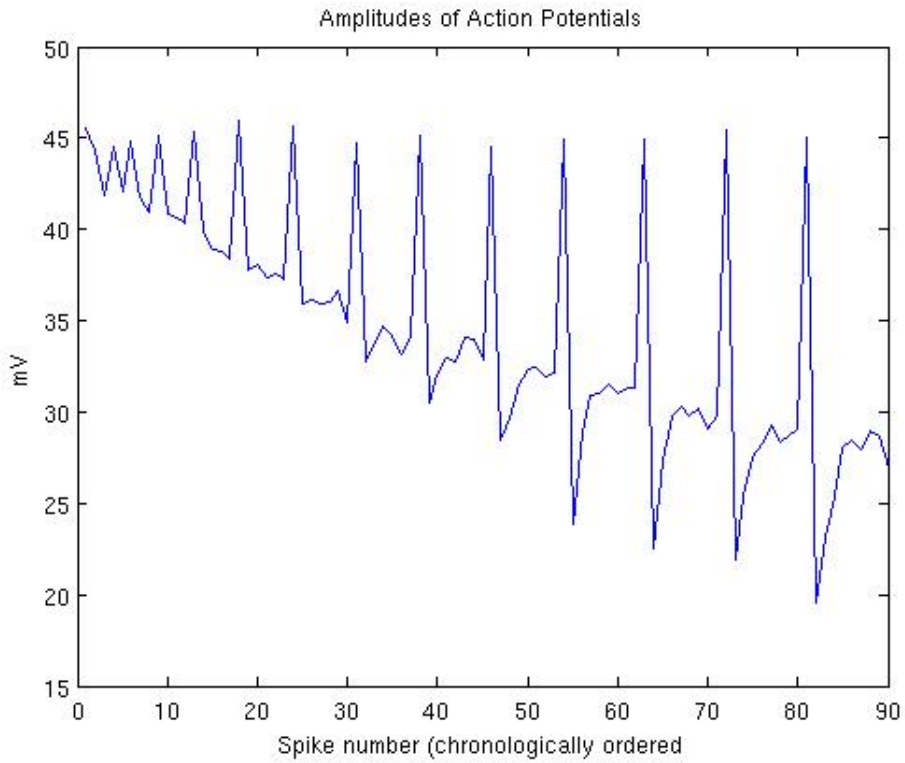
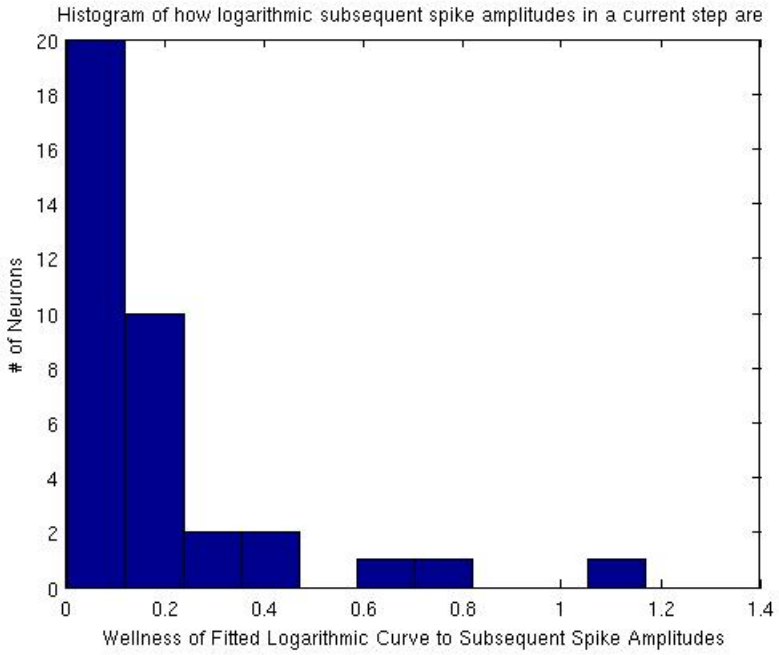


Figure 14: Histogram of Action Potential Amplitude Logarithmicity.



logarithmic curve. (Figure 14)

We can next look at the standard deviations of the action potential amplitudes (averaged standard deviation within-current step). Since dendritic bursting is often represented by a greater fluctuation in spiking amplitudes, this standard deviation might serve as a surrogate marker for whether there was unusual dendritic bursting activity.

Next we look at the average time width of the action potentials at their half-peak. Half-peak was determined by finding the minimum voltage in the last inter-spike interval of that given current step for the purpose of establishing a steady state voltage, and then the half-width voltage was determined by averaging the peak voltage and the steady state. The closest data point to this half-width voltage on either side of the peak were multiplied by the sampling rate to get the time in seconds of the spike duration. Interestingly, the distribution of these spike widths looks bi- or tri-modal, suggesting some natural clustering of this metric.

Since we are also curious about the variance within each current step of the action potential widths (for adaptation and dendritic bursting purposes), we can find the histogram of the standard deviations as well.

Next we want to use the input resistance to convert the minimum current step at which spiking first occurred to a spiking voltage threshold. The distribution of this metric is shown in Figure 18.

This was computed by creating a cell array that matched each consecutive spike (for instance, the y-axis

Figure 15: Histogram of Standard Deviations of Action Potential Amplitudes

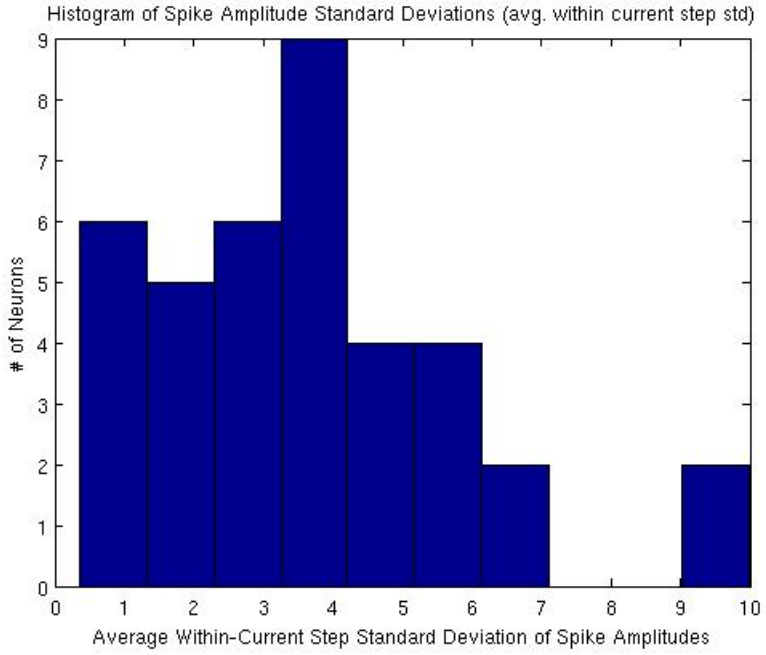


Figure 16: Histogram of Action Potential Width at Half-Peak

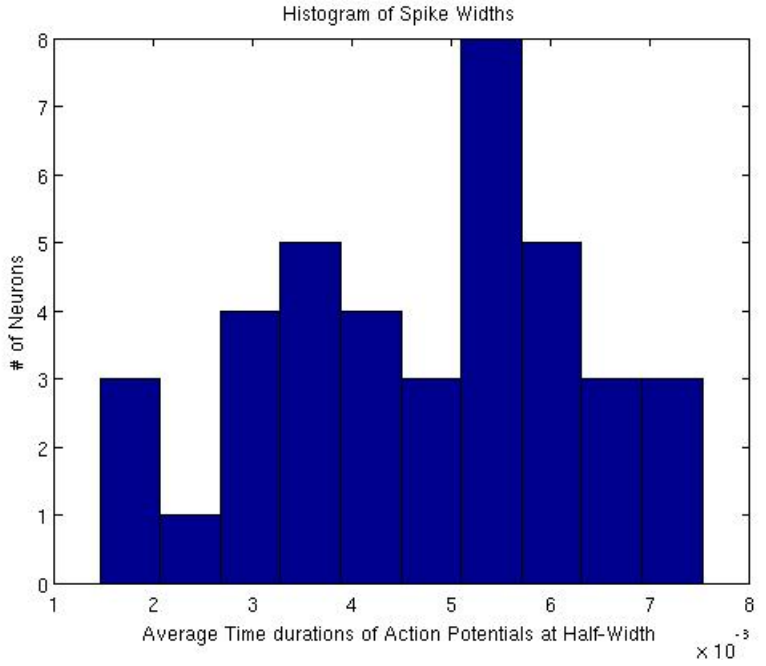
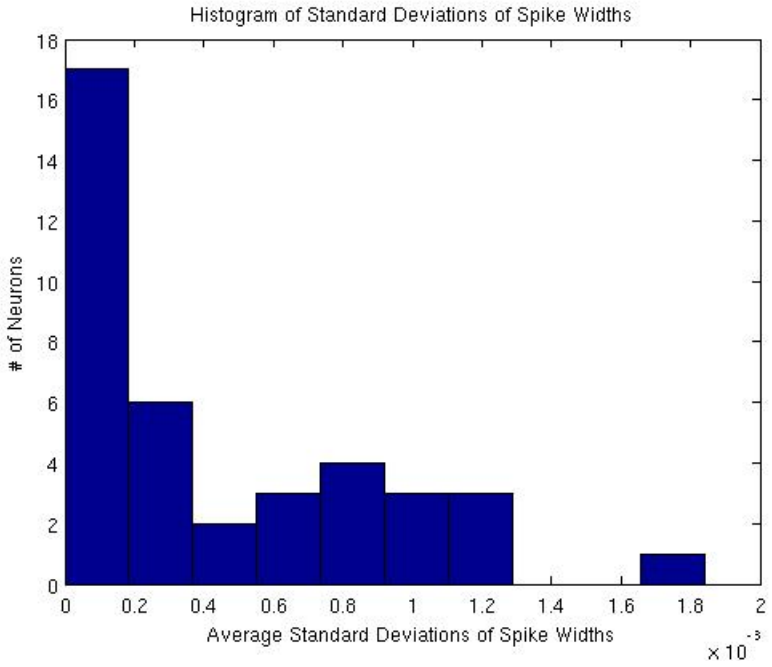


Figure 17: Histogram of Standard Deviations of Action Potential Widths



points from Figure 10) with the current amplitude at which that spike occurred. In this manner, the first current amplitude corresponds with the very first spike a given neuron fired. This current amplitude is normalized to the current baseline - i.e., the current threshold is the current difference in pA from the baseline current at which the neuron began to fire, not the absolute current at which the neuron fired.

Next we can turn our attention to the metrics derived from the idealized spike trains extracted from the data. By far the most used of these metrics is the firing rate. This is perhaps the most informative metric in terms of the electrophysiological classifier, since firing rate encodes brain signals and sensory information, requiring a differentiation by functional type. Interestingly, the distribution of the average firing rate seems to be bi-modal, perhaps corresponding to the sample pool of mostly Regular Spiking cells and about 6 Fast Spiking/Stutterer cells. As an aside, firing rate was calculated by taking the reciprocal of the inter-spike interval.

As a metric that could serve as a surrogate for adaptation, we can look at the variation of firing rate within each current step (and then averaged across all spike-inducing current steps).

Next we want to look at the time between the onset of a spike-inducing current step and the first spike of that current step - i.e., the latency of response. This is an interesting metric, and lately a few papers have shown that some neurons encode information by varying the latency of response after a signal. In the distribution of latencies though, it is difficult to tell if the distribution is multi-modal, or if it is Gaussian with too small of a sample size (since n = 39).

Figure 18: Histogram of Spiking Thresholds

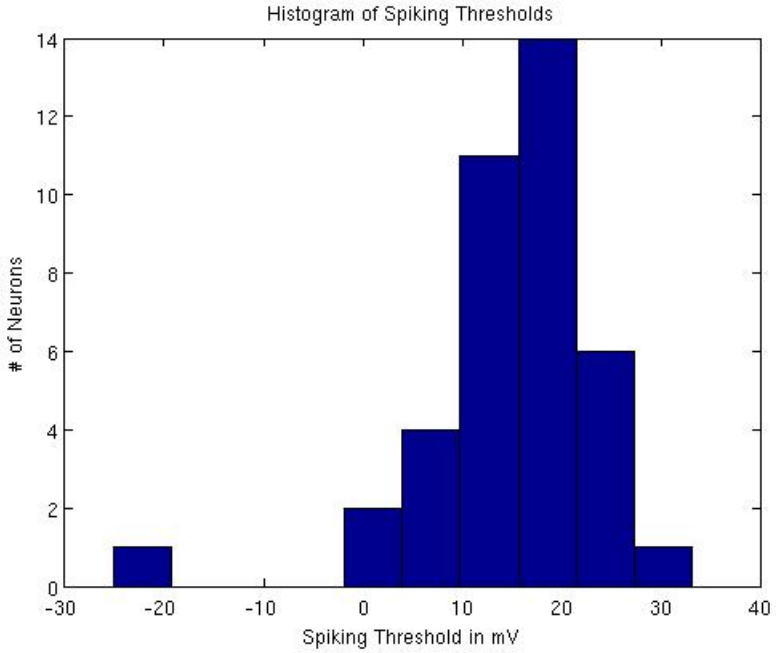


Figure 19: Histogram of Firing Rates

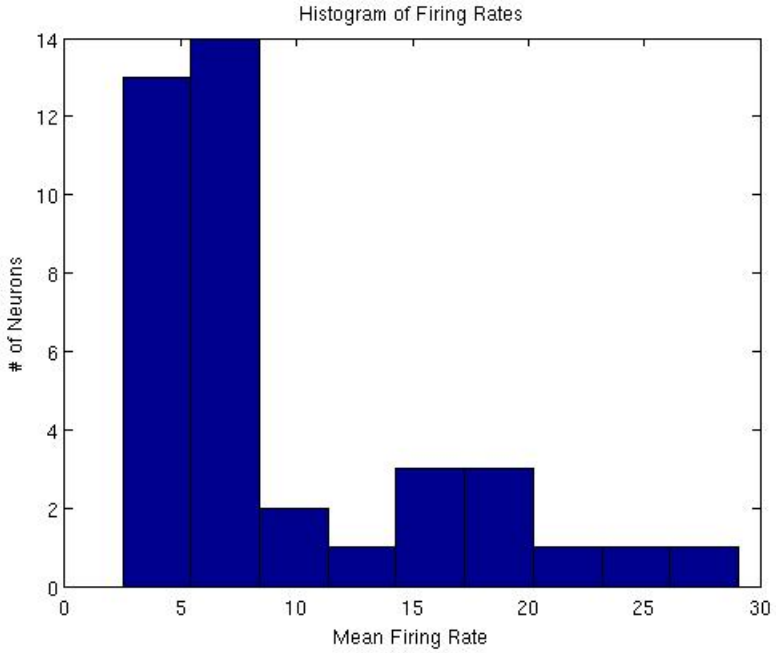
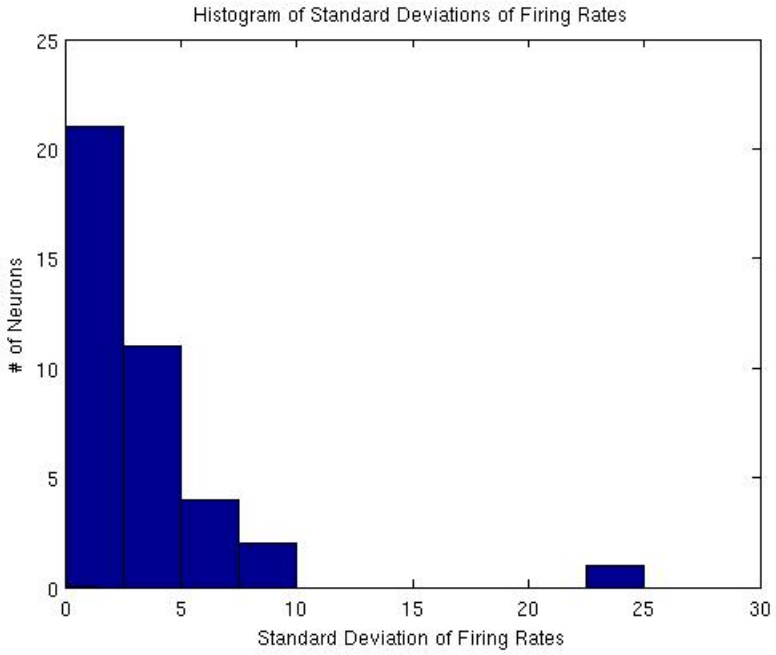


Figure 20: Histogram of Standard Deviations of Firing Rates



Looking at individual neurons, we can see that latencies generally increase as a function of current step, and the curve of latencies over current steps looks somewhat logarithmic. Because of this, I decided to fit the latencies to a logarithmic curve and use the error of this fit as a metric to better capture the structure of the variation in latency.

Interestingly, the distribution of latencies seems to be bimodal - either a cell's latencies will get exponentially shorter as increasingly large current steps are injected, or a cell's latencies will show an irregular pattern of initially fluctuating latencies with an eventual plateauing.

8 Method of Classification (Algorithm)

I implemented a trifecta of classification schemes, first starting with principal component analysis (PCA) to see what kind of dimensionality reduction could be achieved. Next I used a k-means clustering approach to find the optimal number and distribution of classes. Finally I used these k-means clusters as targets for training an artificial feed-forward backpropagation neural network for use in classifying a dataset of 4 unknown neurons.

PCA works by calculating a covariance matrix with zero correlation between its components such that each component has no correlation with any other component. Since the original matrix of metrics will have some correlation between the various metrics, PCA will be useful for its eigenvalues, which allows us to see

Figure 21: Histogram of Average Response Latency

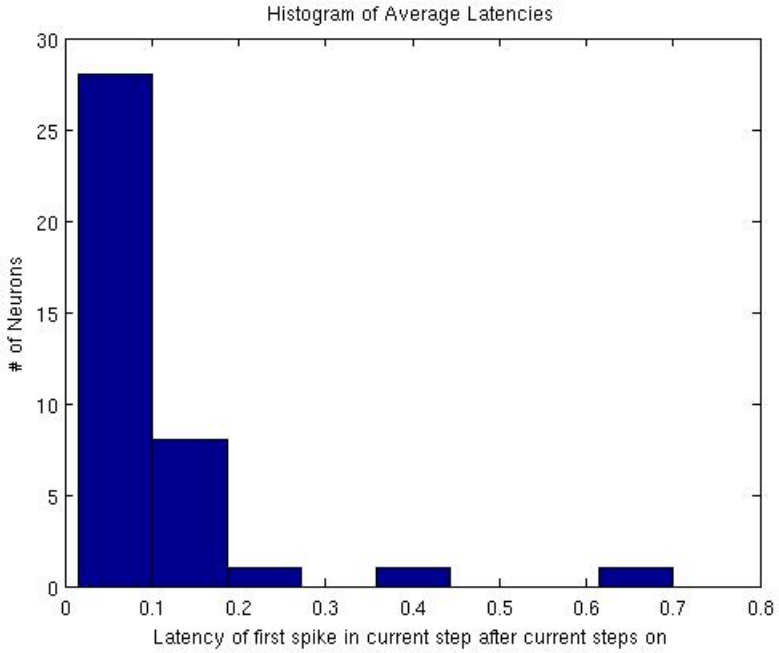
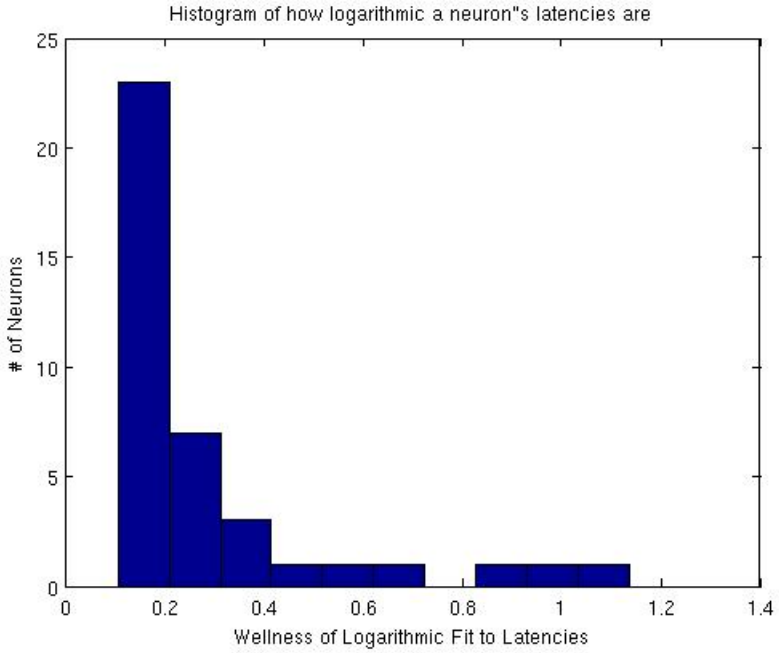


Figure 22: Histogram of Latency Logarithmicity



what percentage of the variance the different principal components explain. This allows us to see how many truly independent parameters there are.

9 Results

9.1 PCA

Below are the eigenvalue results from the principal component analysis. For each eigenvalue I have divided it by the sum of all eigenvalues, resulting in the fraction of variance explained by each respective eigenvalue.

Principal Component	Explanation of Variance
1	0.5998
2	0.3351
3	0.0486
4	0.0145
5	0.0015
6	0.0004
7	0.0001
8	0.0001
9	0.0000
10	0.0000
11	0.0000

It is typical in PCA for the first eigenvalue to explain up to 80% or 90% of the variance. Interestingly, the first principal component only explains 60% of the variance, suggesting that there is a fair independence between several of the hypothetical parameters we examined.

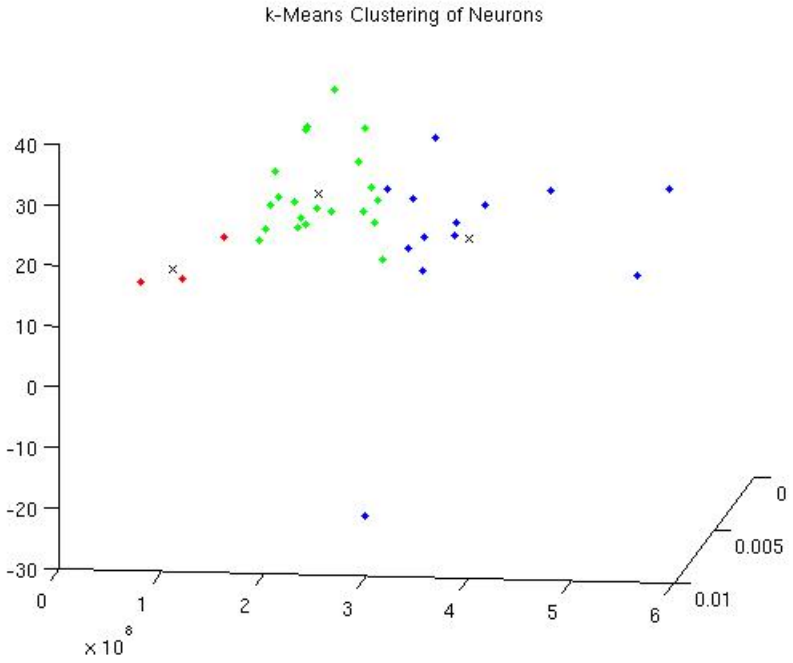
9.2 k-Means Clustering

Using a matrix of 39 neurons by 11 metrics as the input into MATLAB's native *kmeans* command, I experimented with various values of k and compared them to the qualitative descriptions of the spike tests by Peter Kruskal and Joe Kimmel.

My findings are summarized for the first 13 neurons in the following table.

3-means	4-means	Peter's or Joe's Description	Neuron Number	Filename
1	2	RS adapting	1	$20100114_1 2_5 3_1 4.paq$
1	2	RS adapting w/ burst	2	$20100115_1 7_1 2_5 1.paq$
1	2	Very sparse spiking + dendritic burst	3	$20100116_1 4_0 9_0 5.paq$
3	3	Interesting spiking, almost RS	4	$20100116_1 4_1 0_5 8.paq$
1	2	Crazy spiker (adapting)	5	$20100118_1 3_3 8_5 7.paq$
1	2	Crazy spiker (adapting)	6	$20100118_1 3_4 0_1 0.paq$
3	1	FS, maybe stutterer	7	$20100120_1 3_3 4_4 1.paq$
3	1	FS, maybe stutterer	8	$20100126_1 0_1 5_5 7.paq$
1	2	RS adapting (strong adaptation)	9	$20100127_1 6_0 2_1 6.paq$
1	2	RS adapting, high threshold	10	$20100128_1 2_2 6_5 7.paq$
3	1	RS adapting w/ burst	11	$20100201_1 7_1 5_0 5.paq$
1	2	RS adapting	12	$20100202_1 0_3 2_5 3.paq$
1	2	RS adapting	13	$20100202_1 0_3 4_0 0.paq$

Figure 23: k-Means Clustering; k=3

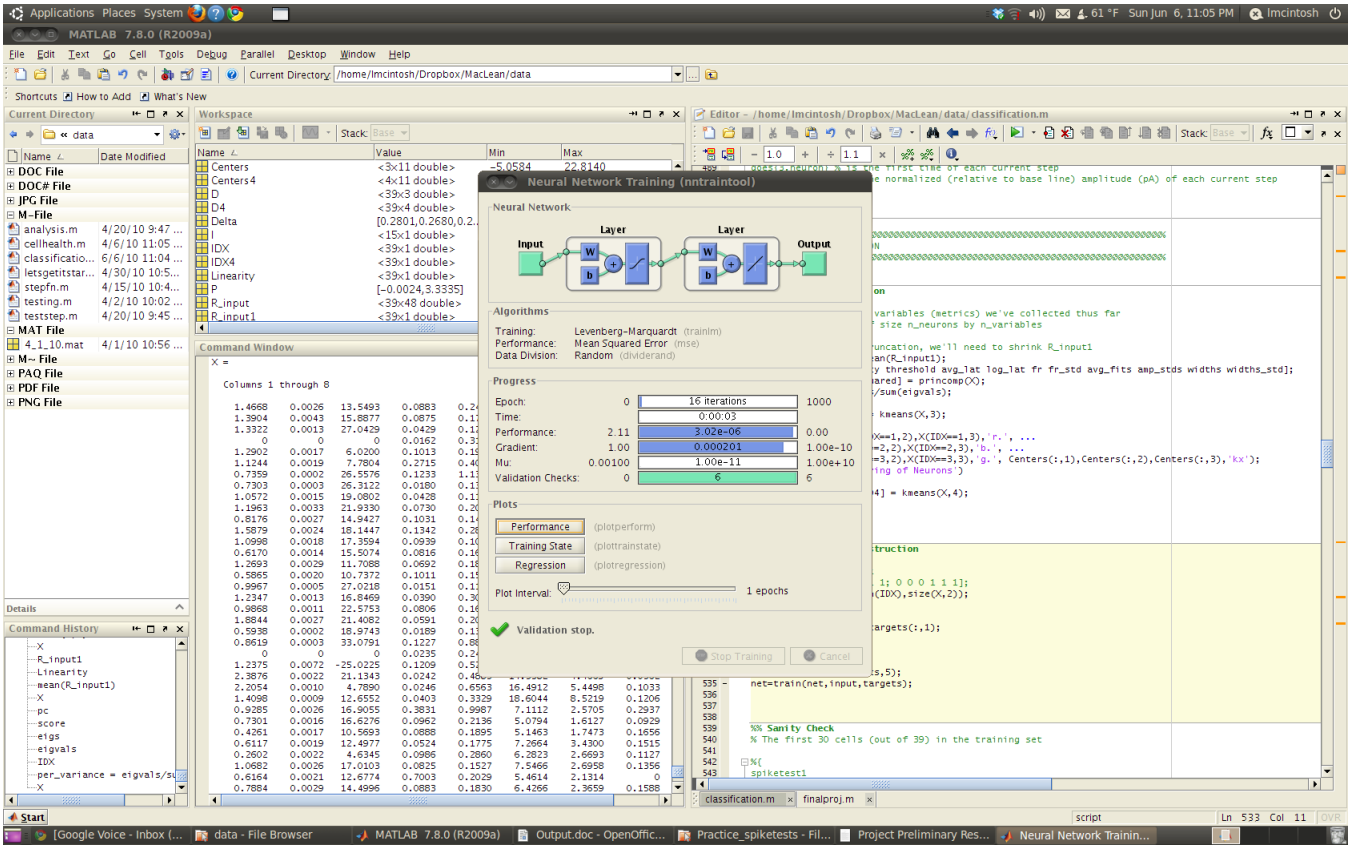


Extrapolating from the entire dataset of all 39 neurons, some patterns become apparent. In the 3-means clustering, "1" corresponds generally to regular spiking (RS) cells, "2" to irregular spiking cells, and "3" to fast spiking (FS)/stutterer cells. However, the clustering is not just dependent on these firing rate observations - there is some segregation among RS cells into RS and irregular categories, particularly when there is unusual bursting activity. There are some neurons also where the qualitative description was unsure of what to make of it (for example, see Neuron 3), but the clustering method sorted it into one of the categories (as an RS cell for instance, even though spiking was sparse). This phenomenon could either be an accurate quantitative classification of the neurons, or an artifact of the number of specified clusters.

Let's look at the 3-D figure of the 3-means clustering (Figure 23). While there is some good clustering, the points are fairly close to each other and boundaries seem somewhat arbitrary.

Next looking at the 4-means clustering, we lose a lot of the strong segregation between FS and RS cells, although category "1" generally seemed to represent cells with no adaptation, "2" with cells with adaptation, "3" with both FS and RS cells that didn't have anything especially exciting in common, and "4" with irregular spikers. Because of this ambiguous 3rd category, I stuck with the 3-means approach for training the artificial neural network, although it is interesting to think that the 4-means clustering was somewhat able to sort out adaptation characteristics.

Figure 24: Screenshot of Artificial Neural Network in Action



9.3 Artificial Neural Network

I next constructed a feedforward, backpropagation artificial neural network with 5 hidden interneuron layers using Matlab's Neural Network Toolbox (Figure 24). I initiated this neural network with tangent sigmoidal hidden layer transfer functions and linear output transfer functions, using the Levenberg-Marquardt algorithm to minimize the neural network functions over the parameters. As a measure of performance I used mean squared error.

Using the classifications issued by the k-means analysis in the step before, I assigned the neuron-specific classification "target" to each one of the 11 metrics, and reiterated for each neuron, such that each one of the metrics associated with a given neuron was also associated with that given neuron's classification. Then, loading a testing dataset of 4 neurons that had not been qualitatively observed, I tested the classification scheme by using the trained neural network to assign each of the 4 neurons to one of the 3 already-established classes. The result of this test was that each of the 4 neurons was assigned to the regular spiking (RS) category. While this was somewhat unexciting, this is not necessarily wrong, since the vast majority of patch-clamped cells are RS cells, and I drew the test cells randomly from the distribution. Looking at the spiking behavior of these test neurons qualitatively, they do indeed bear a similarity to the other RS cells.

Ultimately however, the classification of all neurons based on these 11 metrics resulted in a semi arbitrary classification scheme, suggesting that most of the parameters vary continuously, with the exception of firing rate (and adaptation to a certain extent), which is immediately observable qualitatively. One thing to consider however in future analyses is the extent to which deviations in classification among all cells qualitatively labeled as "RS" represent artifacts or represent true differences (subclasses) within these larger qualitative classes. This could possibly be determined in the future by comparing different classification types (morphological, chemical) to the electrophysiological categories, and seeing if there are physical parameters that help predict the emergence of these subclasses.

10 Code (all 1,100+ lines of it)

```
% classification.m
%% import

clear all;
close all;
addpath /home/lmcintosh/Dropbox/MacLean/
addpath /home/lmcintosh/Dropbox/MacLean/PaqTools
addpath /home/lmcintosh/Dropbox/MacLean/data
addpath /home/lmcintosh/Desktop/Peter_Data_SpikeTests

directory_name = '/home/lmcintosh/Desktop/Peter_Data_SpikeTests';
files = dir(fullfile(directory_name,'*.paq'));
% aside: 20100416_16_10_29.paq is very noisy; best to exclude it

%% store paq object data as voltage and current
%paqobj = paq('/home/lmcintosh/Desktop/Peter_Data_SpikeTests/20100419_17_45_39.paq');
% spike datasets:           % in /Peter_Data_SpikeTests/Practice_spiketests
% 080109-05_17_01PM.paq
% 080709-07_21_16PM.paq
% 20091106_14_42_48.paq
```

```
% 20091111_11_34_25.paq
% 20091113_14_51_42.paq

% 022409-12_47_12PM.paq     % in /data folder
voltage = cell(1,length(files));
current = cell(1,length(files));

for i = 1:length(files);
    paqobj(i) = paq(['/home/lmcintosh/Desktop/Peter_Data_SpikeTests/' files(i).name]);
    traces = paqobj(i).data('channels',1:4,[0,0]);
    % the spike test will have the maximum variance
    if max(var(traces(:,1:2))) > max(var(traces(:,3:4)));
        voltage{:,i} = traces(:,1);
        current{:,i} = traces(:,2);
    else
        voltage{:,i} = traces(:,3);
        current{:,i} = traces(:,4);
    end
    %figure; plot(voltage)
    %figure; plot(current)
end


% Throw out Neuron 4 - cannot calculte R_input because all positive,
% spiking voltage depolarizations (not normal current steps)


%% determine idealized spike trains
close all
% samplingrate = 10000;  % Hz

% each cell of "spiketrains" will be a vector of spike times in seconds
spiketrains = cell(1,length(voltage));
% each cell of "amplitudes" will be a vector of actual voltage peaks of
% the idealized spike train
amplitudes = cell(1,length(voltage));
% each goes{1,neuron} will be 0's and 1's corresponding to the current step,
% each goes{2,neuron} will be the current amplitude of that given step
% each goes{3,neuron} will be the first time of each current step
% each goes{4,neuron} will be the (normalized) current amplitude of each current step
goes = cell(4,length(voltage));
% for each neuron the time range will be different
time = cell(1,length(voltage));

for neuron = 1:length(voltage);
    samplingrate = paqobj(neuron).SampleRate;
    rawtrainV = voltage{:,neuron};
    rawtrainC = current{:,neuron};
    endtime = length(rawtrainV)/samplingrate;
    % create a vector of time in seconds that translates directly to the
    % index number of the voltage and current data
```

```
times = 0:1/samplingrate:(endtime-1/samplingrate);
% to find spikes, first take all time points when voltage was > 0
abovezero = find(rawtrainV > 0);
% initialize variables
numspikes = 1; ends = cell(1,1); starts = cell(1,1); k=1;
% take "mountain climbing" approach to find when spike peaks
for instance = 2:length(abovezero);
    starts{1} = abovezero(1);
    if abovezero(instance)~=abovezero(instance-1)+1;
        if abovezero(instance)~=abovezero(instance-1)+2;
            numspikes = numspikes+1;
            ends{k} = abovezero(instance-1);
            starts{k+1} = abovezero(instance);
            k=k+1;
        end
    end
    ends{k} = abovezero(length(abovezero));
end
amps = zeros(numspikes,1);
spiketrain = zeros(numspikes,1);
for spike = 1:numspikes;
    index = find(rawtrainV(starts{spike}:ends{spike})==max(rawtrainV(starts{spike}:ends{spike})));
    index = starts{spike}+index(1)-1;
    % the actual voltage amplitude of spike at time of idealized spike
    amps(spike) = rawtrainV(index);
    % the time stamp of the idealized spike
    spiketrain(spike) = times(index); % in secs
end
spiketrains{1,neuron} = spiketrain;
amplitudes{1,neuron} = amps;

% normalize the current trace steady state to zero
normalized = rawtrainC - mode(rawtrainC);
abovezero = find(abs(normalized) > 5);
indices = zeros(length(times),1);
indices(2:length(times),1) = find(times) - 1;
gocues = ismember(indices,abovezero);
% clean up single-point noise
for go = 3:length(gocues)-2;
    if gocues(go+1) == 0;
        if gocues(go-1) == 0;
            gocues(go) = 0;
        elseif gocues(go-2) == 0;
            gocues(go) = 0;
        end
    elseif gocues(go+2) == 0;
        if gocues(go-1) == 0;
            gocues(go) = 0;
        end
    end
    if abs(gocues(go+1)) > 0;
```

```
            if abs(gocues(go-1)) > 0;
                gocues(go) = gocues(go-1);
            end
        end
    end
    cues = []; u = 1;
    camp = [];
    for go = 2:length(gocues);
        if gocues(go) > 0;
            if gocues(go-1) == 0;
                cues(u) = times(go);
                if length(gocues) > (go+samplingrate-1);
                    camp(u) = mean(normalized(go:go+samplingrate-1));
                end
                u=u+1;
            end
        end
    end
    goes{1,neuron} = gocues;
    goes{2,neuron} = goes{1,neuron}.*rawtrainC;
    goes{3,neuron} = cues;
    goes{4,neuron} = camp;
    time{1,neuron} = times;
end

% plot(time{1,1},goes{1,1},time{1,1},current{1,1})
% clear abovezero amps ans camp cues ends endtime go gocues index indices normalized rawtrainC rawtrain
clearvars -except spiketrains amplitudes goes time voltage current paqobj
%% Raster plot
figure
hold on
k=0;
for j = 1:length(spiketrains);
    sptimes = spiketrains{j};
    for i = 1:length(sptimes)     %Going through each spike time
        line([sptimes(i) sptimes(i)], [k k+1],'Color','k')  %Create a tick mark at sp time
    end
    k=k+1;
end
%ylim([0 length(spike)])
xlabel('Time (sec)')
ylabel('Neurons')
title('Raster plot of all neurons')



%% determine ISI distributions
close all
isi_dist = cell(length(spiketrains),1);
last_isis = cell(length(spiketrains),1);
```

```
firingrates = zeros(length(spiketrains),17); % 17 chosen empirically
latency = cell(length(spiketrains),1);
curramp = cell(length(spiketrains),1);
for j = 1:length(spiketrains);
    sptimes = spiketrains{j};
    % initialize isi's for this neuron
    isis=[]; h=1;
    rate=[];
    % determine isi's
    for i = 1:length(sptimes)-1;
        isis(h,1) = sptimes(i+1)-sptimes(i);
        rate(h,1) = 1/isis(h,1);
        h=h+1;
    end
    % break down isi's for each current step
    h=1; start=1;
    breaks = find(isis>1);
    for stop = 1:length(breaks);
        isi_dist{j,h} = isis(start:breaks(stop)-1);
        if isfinite(mean(rate(start:breaks(stop)-1)));
            firingrates(j,h) = mean(rate(start:breaks(stop)-1));
        end
        h=h+1;
        start = breaks(stop)+1;
    end
    %figure; plot(isi_dist{j,h-1});
    last_isis{j,1} = isi_dist{j,h-1};

    % determine latency
    cues = goes{3,j}; % PROBLEM WITH GOES{3,28} -> 572 points  -> attempted noise reduction resolution,
    camp = goes{4,j};
    lat = zeros(length(cues),1);
    curr = zeros(length(cues),1);
    for go = 1:length(cues);
        distance = abs(cues(go)-sptimes);
        if min(distance) < 1;
            lat(go) = min(distance);
            if length(camp) >= go;
                curr(go) = camp(go);
            end
        end
    end
    lat = lat(lat~=0);
    curr = curr(curr~=0);
    latency{j,1} = lat;
    curramp{j,1} = curr;
    %figure; plot(latency{j}); title('Latency'); ylabel('Time (sec)')
    % most show an exponential decay in latency, although some (#34) show
    % logarithmic increase in latency, and some peak then decrease (#25)
    % also the incline of the curve varies somewhat
end
```

```
%% determine input resistance
R_input = zeros(length(spiketrains),23);
R_input1 = zeros(length(spiketrains),1);
Linearity = zeros(length(spiketrains),1);
for neuron = 1:length(spiketrains);
    V_ss = mode(voltage{neuron})*10^(-3);
    V_step = zeros(length(goes{4,neuron}),1);
    I = V_step;
    for step = 1:length(goes{4,neuron});
        nospike = find(goes{4,neuron}(goes{4,neuron}(step)==curramp{neuron}));
        if isempty(nospike);
            I(step) = goes{4,neuron}(step)*10^(-12);
            start = int32(goes{3,neuron}(step)*paqobj(neuron).SampleRate);
            ending = int32((goes{3,neuron}(step)+1)*paqobj(neuron).SampleRate);
            V_step(step) = mean(voltage{neuron}(start:ending));
            V_step(step) = V_step(step)*10^(-3);
            R_input(neuron,step) = (V_step(step) - V_ss)/I(step);
            if goes{4,neuron}(step) < 0 && goes{4,neuron}(step+1) > 0;
                R_input1(neuron) = mean(R_input(neuron,step-3:step-1));
            end
        end
    end
    V_step = V_step(V_step~=0);
    I = I(1:length(V_step));
    V_diff = V_step-V_ss;
    %{
    % Plot the V-I relationship
    figure; plot(I,V_diff);
    xlabel('Current')
    ylabel('Voltage difference')
    title(['V-I relationship, Neuron ' num2str(neuron)]);
    %}
    % Determine how linear the V-I relationship is:
    [P,S] = polyfit(I,V_diff,1);
    [Y,Delta] = polyval(P,I,S);
    Linearity(neuron) = mean(Delta);
    Linearity = Linearity(Linearity~=Inf);
    removeme = find(isnan(Linearity));
    for remove = 1:length(removeme);
        Linearity(removeme(remove)) = 0;
    end
end

figure; hist(Linearity);
xlabel('Degree of Linearity')
ylabel('# of Neurons')
title('Histogram of Linearity of V-I relationship for all neurons')

figure; hist(R_input1);
```

```
xlabel('Input Resistances (ohms)')
ylabel('# of Neurons')
title('Histogram of Average Input Resistance')

%% Determine voltage threshold for spiking
threshold = zeros(length(spiketrains),1);
for neuron = 1:length(spiketrains);
    threshold(neuron) = (curramp{neuron}(1)*10^(-12))*R_input1(neuron); % in Volts
    threshold(neuron) = threshold(neuron)*10^3; % convert to mV
end


figure; hist(threshold); xlabel('Spiking Threshold in mV');
ylabel('# of Neurons'); title('Histogram of Spiking Thresholds')

%% Latency metric
nneurons = length(voltage);
r_s = zeros(nneurons,1);
log_lat = r_s;
avg_lat = r_s;
for i = 1:nneurons;
    numlat = length(latency{i,:});
    t = 0:numlat-1;
    [P,S] = polyfit(t,log(latency{i,:})',1);
    [r,p] = corrcoef(P(1)*t-P(2),log(latency{i,:})');
    r_s(i) = r(2);
    [Y,Delta] = polyval(P,t,S);
    log_lat(i) = mean(Delta);
    avg_lat(i) = mean(latency{i});
end

%figure; hist(r_s,10)
figure; hist(log_lat)
xlabel('Wellness of Logarithmic Fit to Latencies')
ylabel('# of Neurons')
title('Histogram of how logarithmic a neuron"s latencies are')

figure; hist(avg_lat,8)
title('Histogram of Average Latencies')
ylabel('# of Neurons')
xlabel('Latency of first spike in current step after current steps on')


%% Firing Rate metrics
fr = zeros(length(spiketrains),1);
fr_std = zeros(length(spiketrains),1);
for neuron = 1:length(spiketrains);
    frs = firingrates(neuron,:);
    frs = frs(frs~=0);
    fr(neuron) = mean(frs);
    fr_std(neuron) = std(frs);
```

```
end

figure; hist(fr,9);
xlabel('Mean Firing Rate'); ylabel('# of Neurons')
title('Histogram of Firing Rates')

figure; hist(fr_std);
xlabel('Standard Deviation of Firing Rates'); ylabel('# of Neurons')
title('Histogram of Standard Deviations of Firing Rates')

%% Spike Amplitude Metrics
% First, we need to add a "current step" dimension to our "amplitudes" cell
% array.
sp_amplitudes = cell(length(spiketrains),1);
spiketrains_perstep = cell(length(spiketrains),1);
avg_fits = zeros(length(spiketrains),1);
amp_stds = zeros(length(spiketrains),1);
for neuron = 1:length(spiketrains);
    fit = zeros(length(goes{4,neuron}));
    stds = zeros(length(goes{4,neuron}));
    i=1;
    for step = 1:length(goes{4,neuron});
        start = goes{3,neuron}(step);
        ending = (goes{3,neuron}(step)+1);
        indices = find(spiketrains{neuron} > start & spiketrains{neuron} < ending);
        if isempty(indices) == 0;
            sp_amplitudes{neuron,i} = amplitudes{neuron}(indices);
            spiketrains_perstep{neuron,i} = spiketrains{neuron}(indices);
            nspikes = length(indices);
            x = 1:nspikes-1; % let's model all spikes excluding the first of each current step
            logs = log(sp_amplitudes{neuron,i})';
            [P,S] = polyfit(x,logs(1:nspikes-1),1);
            [Y,Delta] = polyval(P,x,S);
            onlyfin = isfinite(Delta);
            fit(i) = mean(Delta(onlyfin));
            stds(i) = std(sp_amplitudes{neuron,i});
            i=i+1;
        end
    end
    fit = fit(fit~=0);
    onlyfin = isfinite(fit);
    if isnan(mean(fit(onlyfin)))
        avg_fits(neuron) = 0;
    else
        avg_fits(neuron) = mean(fit(onlyfin));
    end
    stds = stds(stds~=0);
    if isnan(mean(stds))
        amp_stds(neuron) = 0;
    else
        amp_stds(neuron) = mean(stds);
```

```
    end
end

figure; hist(avg_fits);
title('Histogram of how logarithmic subsequent spike amplitudes in a current step are')
ylabel('# of Neurons')
xlabel(' Wellness of Fitted Logarithmic Curve to Subsequent Spike Amplitudes')

figure; hist(amp_stds);
title('Histogram of Spike Amplitude Standard Deviations (avg. within current step std)')
xlabel('Average Within-Current Step Standard Deviation of Spike Amplitudes')
ylabel('# of Neurons')


%% Spike Width
% strategy is to take the width at half voltage between peak and baseline
% where baseline is minimum voltage between last two spikes
widths = zeros(length(isi_dist),1);
widths_std = widths;
for neuron = 1:length(isi_dist);
    spwidth = zeros(size(isi_dist,2),1);
    for step = 1:size(isi_dist,2);
        if isempty(spiketrains_perstep{neuron,step}) == 0;
            nisis = size(isi_dist{neuron,step},1); % nisis will be nspikes-1
            if nisis == 0;
                peak_index = int32(spiketrains_perstep{neuron,step}(1)*paqobj(neuron).SampleRate);
                baseline = min(voltage{neuron}(peak_index:(peak_index+(paqobj(neuron).SampleRate)/2)));
            else
                after = int32(((isi_dist{neuron,step}(nisis))/2)*paqobj(neuron).SampleRate);
                secondtolast = int32(spiketrains_perstep{neuron,step}(length(spiketrains_perstep{neuron
                baseline = min(voltage{neuron}(secondtolast:(secondtolast+after)));
            end
            spwidths = zeros(nisis+1,1);
            spheights = zeros(nisis+1,1);
            stds = zeros(nisis+1,1);
            for i = 1:length(spiketrains_perstep{neuron,step}); % should = nisis+1
                peak_index = int32(spiketrains_perstep{neuron,step}(i)*paqobj(neuron).SampleRate);
                peak = voltage{neuron}(peak_index);
                spheights(i) = (peak+baseline)/2; % note that these are actually voltages, not time wid
                [a firstside] = min(abs(spheights(i)-voltage{neuron}(peak_index-300:peak_index)));
                [a secondside] = min(abs(spheights(i)-voltage{neuron}(peak_index:peak_index+300)));
                spwidths(i) = ((300-firstside)+secondside)/paqobj(neuron).SampleRate; % in seconds
            end
            spwidth(step) = mean(spwidths);
            stds(step) = std(spwidths);
        end
    end
    spwidth = spwidth(spwidth~=0);
    widths(neuron) = mean(spwidth);
    widths_std(neuron) = mean(stds);
end
```

```
figure; hist(widths);
xlabel('Average Time durations of Action Potentials at Half-Width')
ylabel('# of Neurons')
title('Histogram of Spike Widths')

figure; hist(widths_std);
xlabel('Average Standard Deviations of Spike Widths')
ylabel('# of Neurons')
title('Histogram of Standard Deviations of Spike Widths')


%% Important metrics
%{
METRICS:
R_input1(neuron)                % Average Input Resistance across curr. steps
R_input(neuron,current step)    % Each calculated input resistance
Linearity(neuron)               % How linear each V-I relationship is
threshold(neuron)               % spiking threshold in voltage
latency{neuron,current step}    % Time (sec) until first spike after current steps on
avg_lat(neuron)                 % Average latencies
log_lat(neuron)                 % How logarithmic a neuron's latencies are across current steps
fr(neuron)                      % Average firing rate (averaging fr across current steps)
fr_std(neuron)                  % Average standard deviation of firing rates (just looking at 1/ISIs)
                                    % i.e., to capture adaptation
avg_fits(neuron)                % How (on avg.) subsequent spike amplitudes fit a log. curve
amp_stds(neuron)                % Average standard deviation of spike amplitudes within current step
widths(neuron)                  % Time (sec) duration of spike at half-width
widths_std(neuron)              % Average standard deviation of within-current step a.p widths


isi_dist{neuron,current step}  % each row of cells is a different neuron,
                               % column 1 is the first current step that
                               % there was spiking for a given neuron
(last_isis{neuron,1})
latency{neuron,1}       % a vector of latencies (sec) across current steps
amplitudes{1,neuron}    % peak action potential amplitude
firingrates(neuron,:)   % same kind of structure as isi_dist but a matrix
                        % with zeros where there was 1 or no spikes.
                        % Calculated by taking mean(1/isi_i)
curramp{neuron,1}       % each cell is a vector of current amplitudes
                        % corresponding to the current steps associated
                        % with spiking; so the first number is the current
                        % in pA of the first step that caused the neuron to
                        % spike.


OTHER STUFF:
Whole trace:
current{1,neuron}
voltage{1,neuron}
```

```
time{1,neuron}  % for each neuron the time range (secs) will be different

Spike data:
spiketrains{1,neuron}   % vectors of spike times in seconds
amplitudes{1,neuron}    % vectors of voltage amplitudes for each spike

Current steps:
goes{1,neuron} % are 0's and 1's corresponding to the current step,
goes{2,neuron} % are current amplitudes for each given step
goes{3,neuron} % is the first time of each current step
goes{4,neuron} % is the normalized (relative to base line) amplitude (pA) of each current step

%}

%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ACTUAL CLASSIFICATION
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%% Actual Classification

% Let's do PCA on the variables (metrics) we've collected thus far
% Let X be a matrix of size n_neurons by n_variables

% because of Matlab truncation, we'll need to shrink R_input1
R_input1 = R_input1/mean(R_input1);
X = [R_input1 Linearity threshold avg_lat log_lat fr fr_std avg_fits amp_stds widths widths_std];
[pc,score,eigvals,tsquared] = princomp(X);
per_variance = eigvals/sum(eigvals);

[IDX,Centers,SumD,D] = kmeans(X,3);
figure;
plot3(X(IDX==1,1),X(IDX==1,2),X(IDX==1,3),'r.', ...
    X(IDX==2,1),X(IDX==2,2),X(IDX==2,3),'b.', ...
    X(IDX==3,1),X(IDX==3,2),X(IDX==3,3),'g.', Centers(:,1),Centers(:,2),Centers(:,3),'kx');
title('k-Means Clustering of Neurons')

[IDX4,Centers4,SumD4,D4] = kmeans(X,4);


close all

%% Neural Network Construction

% Train neural network
% targets = [0 0 0 1 1 1; 0 0 0 1 1 1];
targets = zeros(length(IDX),size(X,2));
targets(:,1) = IDX';
for var = 2:size(X,2);
    targets(:,var) = targets(:,1);
```

```
end
targets = targets';
input = X';
net=newff(input,targets,5);
net=train(net,input,targets);


clearvars -except X IDX per_variance net paqobj input targets
save afewthings.mat

%% Sanity Check
% The first 30 cells (out of 39) in the training set

%{
spiketest1
20100419_17_45_39.paq
RS adapting

spiketest2
20100419_17_47_17.paq
Rs adapting

spiktest1
20100416_16_10_29.paq
RS small adaptation

spiketest
220100414_12_31_04.paq
FS

spiketest2
20100405_15_15_45.paq
RS little or no adaptation
large AHP

spiketest2
20100405_15_17_20.paq
RS addapting almost FS
maybe a low thresh?
spikes eventually get exhausted

spiketest2
20100327_15_21_12.paq
-noise hump made cell spike
RS addapter
dend burst

spiketest2
20100223_10_32_51.paq
stutt
```

spiketest2 manual
20100223_10_34_38.paq
stutt / FS

spiketest2
20100219_10_31_39.paq
stutt FS

spiketest2
20100216_09_49_29.paq
only spiked twice for a long time
eventually burst

spiketest2
20100213_13_36_01.paq
huge inp res
bursting fs

spiketest2
20100210_12_23_00.paq
maybe LThresh spiker with addapting

spiketest2
20100209_11_53_12.paq
stutterer

spiketest1
20100208_18_24_28.paq
RS addapting

spiketest1
20100206_13_43_52.paq
RS addapting

spiketest1
20100204_15_34_45.paq
RS addapting

spiketest1:
\20100202_10_32_53.paq
regular/adapting.

spiketest2:
20100202_10_34_00.paq
regular/adapting.

spiketest1
20100201_17_15_05.paq
RS addapting dend burst

spiketest1

```
20100128_12_26_57.paq
RS addapting
high thresh,...

spiketest2
20100127_16_02_16.paq
RS addapting (strong addap)

spiketest2
20100126_10_15_57.paq
Stutt FS

spiketest2
20100120_13_34_41.paq
FS maybe stutt

spiketest1
20100118_13_38_57.paq
crazy spiker

spiketest2
20100118_13_40_10.paq
crazy spiker

spiketest2
20100116_14_09_05.paq
very sparse spiking
dendritic burst

manual spiketest
20100116_14_10_58.paq
interesting spiking, not sure if I should call it RS

spiketest2
20100115_17_12_51.paq
RS addapting w/ burst

spiketest1
20100114_12_53_14.paq
RS addapting
bursts
%}

%% Proper testing
close all
addpath /home/lmcintosh/Desktop/Peter_Data_SpikeTests/Practice_spiketests
directory_name = '/home/lmcintosh/Desktop/Peter_Data_SpikeTests/Practice_spiketests';
files_testing = dir(fullfile(directory_name,'*.paq'));

%% store paq object data as voltage and current
%paqobj = paq('/home/lmcintosh/Desktop/Peter_Data_SpikeTests/20100419_17_45_39.paq');
```

```
% spike datasets:           % in /Peter_Data_SpikeTests/Practice_spiketests
% 080109-05_17_01PM.paq
% 080709-07_21_16PM.paq
% 20091106_14_42_48.paq
% 20091111_11_34_25.paq
% 20091113_14_51_42.paq
close all
clear var

% 022409-12_47_12PM.paq     % in /data folder
voltage = cell(1,length(files_testing));
current = cell(1,length(files_testing));

for i = 1:length(files_testing);
    paqobj2(i) = paq(['/home/lmcintosh/Desktop/Peter_Data_SpikeTests/Practice_spiketests/' files_testing
    traces = paqobj2(i).data('channels',1:4,[0,0]);
    % the spike test will have the maximum variance
    if max(var(traces(:,2))) > max(var(traces(:,4)));
        voltage{:,i} = traces(:,1);
        current{:,i} = traces(:,2);
    else
        voltage{:,i} = traces(:,3);
        current{:,i} = traces(:,4);
    end
    figure; plot(voltage{i})
    title(['Test Neuron ' num2str(i)])
    %figure; plot(current)
end


% Throw out Neuron 4 - cannot calculte R_input because all positive,
% spiking voltage depolarizations (not normal current steps)


%% determine idealized spike trains
close all
% samplingrate = 10000;  % Hz

% each cell of "spiketrains" will be a vector of spike times in seconds
spiketrains = cell(1,length(voltage));
% each cell of "amplitudes" will be a vector of actual voltage peaks of
% the idealized spike train
amplitudes = cell(1,length(voltage));
% each goes{1,neuron} will be 0's and 1's corresponding to the current step,
% each goes{2,neuron} will be the current amplitude of that given step
% each goes{3,neuron} will be the first time of each current step
% each goes{4,neuron} will be the (normalized) current amplitude of each current step
goes = cell(4,length(voltage));
% for each neuron the time range will be different
time = cell(1,length(voltage));
```

```
for neuron = 1:length(voltage);
    samplingrate = paqobj(neuron).SampleRate;
    rawtrainV = voltage{:,neuron};
    rawtrainC = current{:,neuron};
    endtime = length(rawtrainV)/samplingrate;
    % create a vector of time in seconds that translates directly to the
    % index number of the voltage and current data
    times = 0:1/samplingrate:(endtime-1/samplingrate);
    % to find spikes, first take all time points when voltage was > 0
    abovezero = find(rawtrainV > 0);
    % initialize variables
    numspikes = 1; ends = cell(1,1); starts = cell(1,1); k=1;
    % take "mountain climbing" approach to find when spike peaks
    for instance = 2:length(abovezero);
        starts{1} = abovezero(1);
        if abovezero(instance)~=abovezero(instance-1)+1;
            if abovezero(instance)~=abovezero(instance-1)+2;
                numspikes = numspikes+1;
                ends{k} = abovezero(instance-1);
                starts{k+1} = abovezero(instance);
                k=k+1;
            end
        end
        ends{k} = abovezero(length(abovezero));
    end
    amps = zeros(numspikes,1);
    spiketrain = zeros(numspikes,1);
    for spike = 1:numspikes;
        index = find(rawtrainV(starts{spike}:ends{spike})==max(rawtrainV(starts{spike}:ends{spike})));
        index = starts{spike}+index(1)-1;
        % the actual voltage amplitude of spike at time of idealized spike
        amps(spike) = rawtrainV(index);
        % the time stamp of the idealized spike
        spiketrain(spike) = times(index); % in secs
    end
    spiketrains{1,neuron} = spiketrain;
    amplitudes{1,neuron} = amps;

    % normalize the current trace steady state to zero
    normalized = rawtrainC - mode(rawtrainC);
    abovezero = find(abs(normalized) > 5);
    indices = zeros(length(times),1);
    indices(2:length(times),1) = find(times) - 1;
    gocues = ismember(indices,abovezero);
    % clean up single-point noise
    for go = 3:length(gocues)-2;
        if gocues(go+1) == 0;
            if gocues(go-1) == 0;
                gocues(go) = 0;
            elseif gocues(go-2) == 0;
                gocues(go) = 0;
```

```
            end
        elseif gocues(go+2) == 0;
            if gocues(go-1) == 0;
                gocues(go) = 0;
            end
        end
        if abs(gocues(go+1)) > 0;
            if abs(gocues(go-1)) > 0;
                gocues(go) = gocues(go-1);
            end
        end
    end
    cues = []; u = 1;
    camp = [];
    for go = 2:length(gocues);
        if gocues(go) > 0;
            if gocues(go-1) == 0;
                cues(u) = times(go);
                if length(gocues) > (go+samplingrate-1);
                    camp(u) = mean(normalized(go:go+samplingrate-1));
                end
                u=u+1;
            end
        end
    end
    goes{1,neuron} = gocues;
    goes{2,neuron} = goes{1,neuron}.*rawtrainC;
    goes{3,neuron} = cues;
    goes{4,neuron} = camp;
    time{1,neuron} = times;
end

% plot(time{1,1},goes{1,1},time{1,1},current{1,1})
% clear abovezero amps ans camp cues ends endtime go gocues index indices normalized rawtrainC rawtrain
clearvars -except spiketrains amplitudes goes time voltage current paqobj
%% Raster plot
figure
hold on
k=0;
for j = 1:length(spiketrains);
    sptimes = spiketrains{j};
    for i = 1:length(sptimes)     %Going through each spike time
        line([sptimes(i) sptimes(i)], [k k+1],'Color','k')  %Create a tick mark at sp time
    end
    k=k+1;
end
%ylim([0 length(spike)])
xlabel('Time (sec)')
ylabel('Neurons')
title('Raster plot of all neurons')
```

```
%% determine ISI distributions
close all
isi_dist = cell(length(spiketrains),1);
last_isis = cell(length(spiketrains),1);
firingrates = zeros(length(spiketrains),17); % 17 chosen empirically
latency = cell(length(spiketrains),1);
curramp = cell(length(spiketrains),1);
for j = 1:length(spiketrains);
    sptimes = spiketrains{j};
    % initialize isi's for this neuron
    isis=[]; h=1;
    rate=[];
    % determine isi's
    for i = 1:length(sptimes)-1;
        isis(h,1) = sptimes(i+1)-sptimes(i);
        rate(h,1) = 1/isis(h,1);
        h=h+1;
    end
    % break down isi's for each current step
    h=1; start=1;
    breaks = find(isis>1);
    for stop = 1:length(breaks);
        isi_dist{j,h} = isis(start:breaks(stop)-1);
        if isfinite(mean(rate(start:breaks(stop)-1)));
            firingrates(j,h) = mean(rate(start:breaks(stop)-1));
        end
        h=h+1;
        start = breaks(stop)+1;
    end
    %figure; plot(isi_dist{j,h-1});
    last_isis{j,1} = isi_dist{j,h-1};

    % determine latency
    cues = goes{3,j}; % PROBLEM WITH GOES{3,28} -> 572 points  -> attempted noise reduction resolution,
    camp = goes{4,j};
    lat = zeros(length(cues),1);
    curr = zeros(length(cues),1);
    for go = 1:length(cues);
        distance = abs(cues(go)-sptimes);
        if min(distance) < 1;
            lat(go) = min(distance);
            if length(camp) >= go;
                curr(go) = camp(go);
            end
        end
    end
    lat = lat(lat~=0);
    curr = curr(curr~=0);
```

```
    latency{j,1} = lat;
    curramp{j,1} = curr;
    %figure; plot(latency{j}); title('Latency'); ylabel('Time (sec)')
    % most show an exponential decay in latency, although some (#34) show
    % logarithmic increase in latency, and some peak then decrease (#25)
    % also the incline of the curve varies somewhat
end


%% determine input resistance
R_input = zeros(length(spiketrains),23);
R_input1 = zeros(length(spiketrains),1);
Linearity = zeros(length(spiketrains),1);
for neuron = 1:length(spiketrains);
    V_ss = mode(voltage{neuron})*10^(-3);
    V_step = zeros(length(goes{4,neuron}),1);
    I = V_step;
    for step = 1:length(goes{4,neuron});
        nospike = find(goes{4,neuron}(goes{4,neuron}(step)==curramp{neuron}));
        if isempty(nospike);
            I(step) = goes{4,neuron}(step)*10^(-12);
            start = int32(goes{3,neuron}(step)*paqobj(neuron).SampleRate);
            ending = int32((goes{3,neuron}(step)+1)*paqobj(neuron).SampleRate);
            V_step(step) = mean(voltage{neuron}(start:ending));
            V_step(step) = V_step(step)*10^(-3);
            R_input(neuron,step) = (V_step(step) - V_ss)/I(step);
            if goes{4,neuron}(step) < 0 && goes{4,neuron}(step+1) > 0;
                R_input1(neuron) = mean(R_input(neuron,step-3:step-1));
            end
        end
    end
    V_step = V_step(V_step~=0);
    I = I(1:length(V_step));
    V_diff = V_step-V_ss;
    %{
    % Plot the V-I relationship
    figure; plot(I,V_diff);
    xlabel('Current')
    ylabel('Voltage difference')
    title(['V-I relationship, Neuron ' num2str(neuron)]);
    %}
    % Determine how linear the V-I relationship is:
    [P,S] = polyfit(I,V_diff,1);
    [Y,Delta] = polyval(P,I,S);
    Linearity(neuron) = mean(Delta);
    Linearity = Linearity(Linearity~=Inf);
    removeme = find(isnan(Linearity));
    for remove = 1:length(removeme);
        Linearity(removeme(remove)) = 0;
    end
end
```

```
figure; hist(Linearity);
xlabel('Degree of Linearity')
ylabel('# of Neurons')
title('Histogram of Linearity of V-I relationship for all neurons')

figure; hist(R_input1);
xlabel('Input Resistances (ohms)')
ylabel('# of Neurons')
title('Histogram of Average Input Resistance')

%% Determine voltage threshold for spiking
threshold = zeros(length(spiketrains),1);
for neuron = 1:length(spiketrains);
    threshold(neuron) = (curramp{neuron}(1)*10^(-12))*R_input1(neuron); % in Volts
    threshold(neuron) = threshold(neuron)*10^3; % convert to mV
end


figure; hist(threshold); xlabel('Spiking Threshold in mV');
ylabel('# of Neurons'); title('Histogram of Spiking Thresholds')

%% Latency metric
nneurons = length(voltage);
r_s = zeros(nneurons,1);
log_lat = r_s;
avg_lat = r_s;
for i = 1:nneurons;
    numlat = length(latency{i,:});
    t = 0:numlat-1;
    [P,S] = polyfit(t,log(latency{i,:})',1);
    [r,p] = corrcoef(P(1)*t-P(2),log(latency{i,:})');
    r_s(i) = r(2);
    [Y,Delta] = polyval(P,t,S);
    log_lat(i) = mean(Delta);
    avg_lat(i) = mean(latency{i});
end

%figure; hist(r_s,10)
figure; hist(log_lat)
xlabel('Wellness of Logarithmic Fit to Latencies')
ylabel('# of Neurons')
title('Histogram of how logarithmic a neuron"s latencies are')

figure; hist(avg_lat,8)
title('Histogram of Average Latencies')
ylabel('# of Neurons')
xlabel('Latency of first spike in current step after current steps on')


%% Firing Rate metrics
```

```
fr = zeros(length(spiketrains),1);
fr_std = zeros(length(spiketrains),1);
for neuron = 1:length(spiketrains);
    frs = firingrates(neuron,:);
    frs = frs(frs~=0);
    fr(neuron) = mean(frs);
    fr_std(neuron) = std(frs);
end

figure; hist(fr,9);
xlabel('Mean Firing Rate'); ylabel('# of Neurons')
title('Histogram of Firing Rates')

figure; hist(fr_std);
xlabel('Standard Deviation of Firing Rates'); ylabel('# of Neurons')
title('Histogram of Standard Deviations of Firing Rates')

%% Spike Amplitude Metrics
% First, we need to add a "current step" dimension to our "amplitudes" cell
% array.
sp_amplitudes = cell(length(spiketrains),1);
spiketrains_perstep = cell(length(spiketrains),1);
avg_fits = zeros(length(spiketrains),1);
amp_stds = zeros(length(spiketrains),1);
for neuron = 1:length(spiketrains);
    fit = zeros(length(goes{4,neuron}));
    stds = zeros(length(goes{4,neuron}));
    i=1;
    for step = 1:length(goes{4,neuron});
        start = goes{3,neuron}(step);
        ending = (goes{3,neuron}(step)+1);
        indices = find(spiketrains{neuron} > start & spiketrains{neuron} < ending);
        if isempty(indices) == 0;
            sp_amplitudes{neuron,i} = amplitudes{neuron}(indices);
            spiketrains_perstep{neuron,i} = spiketrains{neuron}(indices);
            nspikes = length(indices);
            x = 1:nspikes-1; % let's model all spikes excluding the first of each current step
            logs = log(sp_amplitudes{neuron,i})';
            [P,S] = polyfit(x,logs(1:nspikes-1),1);
            [Y,Delta] = polyval(P,x,S);
            onlyfin = isfinite(Delta);
            fit(i) = mean(Delta(onlyfin));
            stds(i) = std(sp_amplitudes{neuron,i});
            i=i+1;
        end
    end
    fit = fit(fit~=0);
    onlyfin = isfinite(fit);
    if isnan(mean(fit(onlyfin)))
        avg_fits(neuron) = 0;
    else
```

```
        avg_fits(neuron) = mean(fit(onlyfin));
    end
    stds = stds(stds~=0);
    if isnan(mean(stds))
        amp_stds(neuron) = 0;
    else
        amp_stds(neuron) = mean(stds);
    end
end

figure; hist(avg_fits);
title('Histogram of how logarithmic subsequent spike amplitudes in a current step are')
ylabel('# of Neurons')
xlabel(' Wellness of Fitted Logarithmic Curve to Subsequent Spike Amplitudes')

figure; hist(amp_stds);
title('Histogram of Spike Amplitude Standard Deviations (avg. within current step std)')
xlabel('Average Within-Current Step Standard Deviation of Spike Amplitudes')
ylabel('# of Neurons')


%% Spike Width
% strategy is to take the width at half voltage between peak and baseline
% where baseline is minimum voltage between last two spikes
widths = zeros(size(isi_dist,1),1);
widths_std = widths;
for neuron = 1:size(isi_dist,1);
    spwidth = zeros(size(isi_dist,2),1);
    for step = 1:size(isi_dist,2);
        if isempty(spiketrains_perstep{neuron,step}) == 0;
            nisis = size(isi_dist{neuron,step},1); % nisis will be nspikes-1
            if nisis == 0;
                peak_index = int32(spiketrains_perstep{neuron,step}(1)*paqobj(neuron).SampleRate);
                baseline = min(voltage{neuron}(peak_index:(peak_index+(paqobj(neuron).SampleRate)/2)));
            else
                after = int32(((isi_dist{neuron,step}(nisis))/2)*paqobj(neuron).SampleRate);
                secondtolast = int32(spiketrains_perstep{neuron,step}(length(spiketrains_perstep{neuron
                baseline = min(voltage{neuron}(secondtolast:(secondtolast+after)));
            end
            spwidths = zeros(nisis+1,1);
            spheights = zeros(nisis+1,1);
            stds = zeros(nisis+1,1);
            for i = 1:length(spiketrains_perstep{neuron,step}); % should = nisis+1
                peak_index = int32(spiketrains_perstep{neuron,step}(i)*paqobj(neuron).SampleRate);
                peak = voltage{neuron}(peak_index);
                spheights(i) = (peak+baseline)/2; % note that these are actually voltages, not time wid
                [a firstside] = min(abs(spheights(i)-voltage{neuron}(peak_index-300:peak_index)));
                [a secondside] = min(abs(spheights(i)-voltage{neuron}(peak_index:peak_index+300)));
                spwidths(i) = ((300-firstside)+secondside)/paqobj(neuron).SampleRate; % in seconds
            end
            spwidth(step) = mean(spwidths);
```

```
            stds(step) = std(spwidths);
        end
    end
    spwidth = spwidth(spwidth~=0);
    widths(neuron) = mean(spwidth);
    widths_std(neuron) = mean(stds);
end

figure; hist(widths);
xlabel('Average Time durations of Action Potentials at Half-Width')
ylabel('# of Neurons')
title('Histogram of Spike Widths')

figure; hist(widths_std);
xlabel('Average Standard Deviations of Spike Widths')
ylabel('# of Neurons')
title('Histogram of Standard Deviations of Spike Widths')


%% Neural Network Classification
load afewthings.mat;
% Let's do PCA on the variables (metrics) we've collected thus far
% Let X be a matrix of size n_neurons by n_variables

% because of Matlab truncation, we'll need to shrink R_input1
R_input1 = R_input1/mean(R_input1);
X_test = [R_input1 Linearity threshold avg_lat log_lat fr fr_std avg_fits amp_stds widths widths_std];
[pc,score,eigvals,tsquared] = princomp(X);
per_variance_test = eigvals/sum(eigvals);

[IDX_test,Centers,SumD,D] = kmeans(X_test,3);
figure;
plot3(X(IDX==1,1),X(IDX==1,2),X(IDX==1,3),'r.', ...
    X(IDX==2,1),X(IDX==2,2),X(IDX==2,3),'b.', ...
    X(IDX==3,1),X(IDX==3,2),X(IDX==3,3),'g.', Centers(:,1),Centers(:,2),Centers(:,3),'kx');
title('k-Means Clustering of Neurons')



% Neural Network Classification
input_testing = X_test';

trainOutputs = sim(net,X');
testOutputs = sim(net,input_testing);
```