Characterization of Nonlinear Neuron Responses

Final Report

Matt Whiteway
Department of Applied Mathematics and Scientific Computing
whit8022@umd.edu

Advisor
Dr. Daniel A. Butts
Neuroscience and Cognitive Science Program
Department of Applied Mathematics and Scientific Computing
Biologicial Sciences Graduate Program
dab@umd.edu

May 19, 2014

Abstract

A common approach to characterizing a sensory neuron's response to stimuli is to use a probabilistic modeling approach. These models use both the stimulus and the neuron's response to the stimulus to estimate the probability of the neuron spiking. This project will investigate some of the well known approaches, starting with simple linear models and progressing to more complex nonlinear models. All models considered use the idea of a "linear receptive field" to characterize a feature subspace of the stimulus on which the neuron's response is dependent. Both moment-based estimators and maximum-likelihood estimators will be used for fitting parameters, dependent upon the model.

Contents

1 Introduction

A fundamental area of interest in the study of single neuron computation is the relationship between the stimuli and the neural response. There are two distinct approaches to this problem, one motivated by the actual biological processes of a neuron and the other motivated by a functional representation of the neuron's behavior.

The biological models follow more of a bottom-up paradigm, modeling intrinsic biological processes that produce the neuron's behavior. Some of the earliest work on neuron modeling followed this approach, from Lapique's integrate-and-fire model (1907) [1] to Hodgkin and Huxley's set of nonlinear differential equations that describe ion channels and voltage differences across the cell membrane (1952) [2]. More recent approaches have utilized a probabalistic modeling framework. This top-down approach does not attempt to capture the reality of how a neuron processes data, but instead tries to faithfully reproduce the functional behavior of a neuron.

Biological models have several inherent issues that prevent their practical implementations, which motivated the development of probabilistic models. Biological models contain many parameters that govern the underlying physical processes, such as ion channel conductances. These parameters are often difficult or impossible to measure experimentally, and their estimation is not much easier given the nonlinear and noisy nature of neurons. Another drawback to biological models is that they tend to be deterministic models; with any deterministic model we can of course introduce noise, but again this requires knowledge of the source or the structure of the noise.

Since techniques in neuroscience are now able to supply us with increasingly detailed physiological data, and since the nervous system itself is probabalistic, the statistical description of a neuron seems like a natural avenue of exploration [6]. Many of these models are parameterized (including some detailed in this paper), but advances in modeling over the past decade have led to robust and efficient schemes for estimating the parameters of these probabilistic models, which has given them an edge over their biological counterparts.

This paper follows the development of these probabilistic models. It details both linear and nonlinear models, and introduces two of the techniques commonly used to estimate the parameters of the given models.

2 Linear Models

Before considering the specific models investigated it will be useful to explain a basic linear model widely used throughout the neuroscience field. The goal of the linear model is to represent the selectivity of the neuron's response to particular features of the stimulus. This selectivity means that certain regions of the stimulus space, called the feature subspace or the receptive field, are more important than others in determining the resulting action of the neuron. The linear models operate by projecting the stimulus onto these lower dimensional subspaces, then subsequently mapping this projection nonlinearly into a firing rate. This firing rate is interpreted as a rate parameter for an inhomogeneous Poisson process that will then give the probability of the neuron spiking [6].

For the remainder of the paper we will assume that we are modeling a sensory neuron in the visual cortex. The stimulus is a single grayscale pixel that stays the same value for a fixed time interval Δt, then changes value instantaneously. This stimulus is represented by a stimulus vector $\mathbf{s}(t)$, where each component is the pixel's value for a time duration of Δt seconds. The response data is a list of times that the neuron spiked during presentation of the stimulus.

It should be noted than when describing this model as "linear", we are refering to the manner in which the stimulus is processed before a nonlinear function maps this into the rate parameter. The form of the linear model described above is called the Linear-Nonlinear-Poisson (LNP) model, and is given by the equation

$$r(t) = F(\mathbf{k} \cdot \mathbf{s}(t)) \tag{1}$$

where $\mathbf{k}$ is the linear receptive field, $\mathbf{s}(t)$ is the stimulus, F is a nonlinear function, and $r(t)$ is the resulting rate parameter. The linear receptive field $\mathbf{k}$ is what we wish to find, $\mathbf{s}(t)$ is a portion of the stimulus whose size depends on the size of $\mathbf{k}$, and F is generally a sigmoidal function that ensures the firing rate is not negative. The following sections will develop different ways in which to estimate $\mathbf{k}$ and F, the unknowns quantities of this equation.

3 Moment Based Estimators

Moment based estimators will be the first technique we consider for finding parameters in the above model. The general idea is that the stimulus is a group of points in stimulus space; the stimuli that elicited spikes are a separate group of points in this same stimulus space, and we want to describe the ways in which these two groups differ. The first approach, the Spike-Triggered Averge (STA), will look for a difference in the means of these groups, or the difference in the first moment. The second approach expands upon the STA by looking at the difference in the second moment, and is hence called the Spike-Triggered Covariance (STC).

3.1 The Spike-Triggered Average

The first way in which we will be estimating the parameters of the LNP is through a technique called the Spike-Triggered Average (STA). The STA assumes that the neuron's response is completely determined by the stimulus presented during a predetermined time interval in the past, and is defined as the mean stimulus that elicited a spike [4]:

$$STA = \frac{1}{N} \sum_{n=1}^{N} \mathbf{s}(t_n) \tag{2}$$

where N is the number of spikes elicited by the stimulus and $\mathbf{s}(t_n)$ is the stimulus that elicited the n^{th} spike. Defined in this way, the STA is a linear receptive field that the neuron is responsive to, and filtering the stimulus through the STA projects the stimulus onto a lower-dimensional subspace. As long as the input stimulus is spherically symmetric (values in each dimension have a distribution with zero mean), we can use the STA as the estimate for the linear filter $\mathbf{k}$ in the LNP model [4].

Now that the filter has been determined all that is left is to estimate the nonlinearity F. In theory we can choose a parametric form of F and fit the parameters using the given data; however, in practice an easier solution is sought. The literature commonly uses what is known as the "histogram method", which essentially creates a discretized version of F[3]. The input space of $\mathbf{k} \cdot \mathbf{s}(t)$ is divided into bins and the average spike count of each bin is calculated using $\mathbf{k}$, $\mathbf{s}(t)$ and the known spike times. In this way we recover a function that has a single value for each bin, and new data can be tested on the model by filtering the new stimulus with $\mathbf{k}$ and using the resulting bin value to estimate the firing rate.

3.1.1 STA Implementation

The implementation of the STA is a relatively straightforward process. What needs to be taken into account is the data that is available to work with: the stimulus vector, and a list of observed spike times. To make the implementation easier the first task is to transform the spike time list into a vector that is the same size as the stimulus vector, with each element holding the number of spikes observed during that time interval. In practice this vector is mostly zeros, with some entries set to one and very few entries set to two or more.

The result of the STA algorithm is shown in figure 1(a) for a filter size of 20 time steps. One aspect of implementing this algorithm that needs to be considered is how far back in the stimulus vector to look at each spike. We can see that there are stimulus features going back to about 15 time steps, and then the recovered filter settles down to zero. This is precisely because the data we are working with follows a Gaussian distribution with zero mean. Here we are seeing that the neuron is no longer detecting stimulus features, and instead we are seeing the average of this Gaussian noise. Hence by inspection we can choose the filter size to be 15 time steps, which will be used for the filter size in the remainder of the paper.

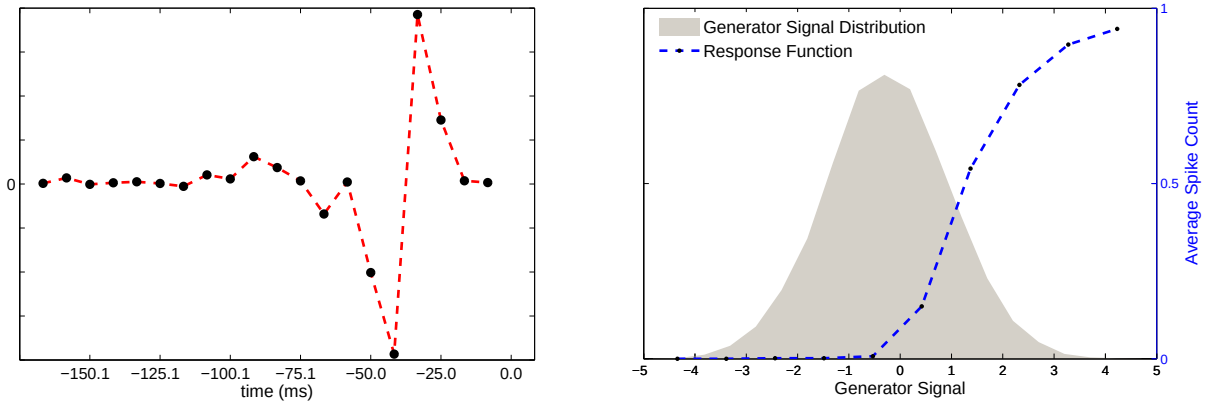

Figure 1: Left: The recovered filter using the STA algorithm for a filter size of 20 time steps. Right: Plotting the nonlinear response function on top of the generator signal distribution shows the relation between the two, which is more important than the particular values each takes. This response function is for a filter of length 15 time steps and an upsampling factor of 1.

The filter that is recovered by the STA has a resolution that is restricted by the frequency of the stimulus change. If we want a higher resolution in the filter we need to sample the stimulus more often, say by a factor of n. The stimulus filter will grow from it's original size $|\mathbf{s}|$ to a size of $n * |\mathbf{s}|$, and the time interval between samplings will decrease by a factor of n.

If the stimulus is then upsampled by a factor $n \geq 1$ the resolution of the filter will be increased accordingly, as shown in figure 2(b).

As mentioned previously, the common approach to modeling the nonlinear response function F in the literature is to use the "histogram method", which bins the values of the generator signal $(\mathbf{k} \cdot \mathbf{s}(t))$ and finds the average spike count for each bin. The result is a discretized version of the response function; given a certain signal $\mathbf{s}'(t)$, we can compute $\mathbf{k} \cdot \mathbf{s}'(t)$, find which bin this value

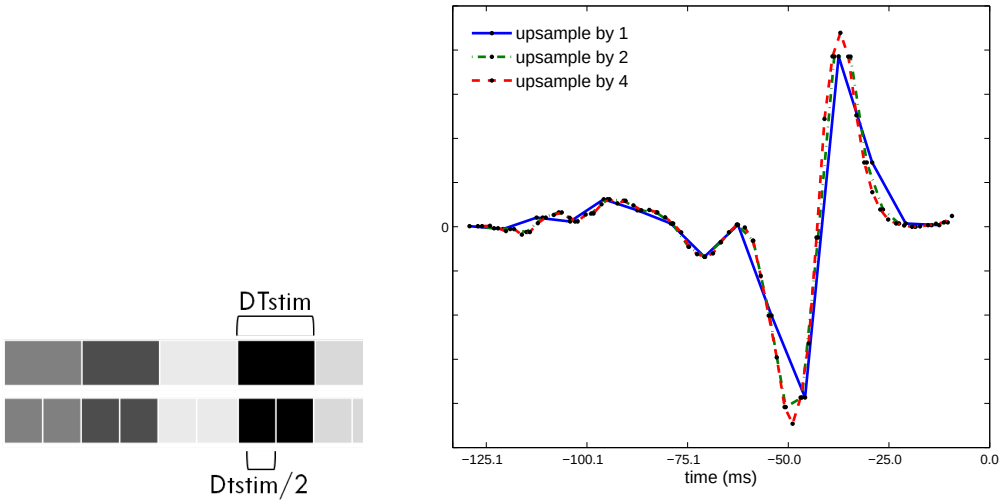

Figure 2: Left: Upsampling the stimulus vector by a factor of 2. Right: Upsampling the stimulus vector produces a filter with higher resolution. Filters shown are 15 original time steps long.

belongs to, and F will return the average spike count we can expect from that particular signal.

Notice that the actual value of the generator signal is not of the utmost importance. We can scale the generator signal by some factor, and the nonlinear function will change accordingly. Instead what we are interested in is the value of the nonlinear response function relative to the distribution of the generator signal; for this reason it is most instructive to look at the response function overlaying the generator signal distribution, as shown in figure 1(b).

3.1.2 STA Validation

The STA algorithm has two components that need to be validated; the first is the implementation of the routine that estimates the filter, and the second is the implementation of the routine that estimates the nonlinear response function.

In order to validate the recovery of the filter, it suffices to check that the program properly locates the spikes and averages the stimulus that precedes each spike during a certain temporal window. We first populate a stimulus vector with 15000 time samples drawn from a Gaussian distribution. We then create an artificial filter that is 10 time steps long, and insert this in place of the Gaussian white noise at random points throughout the stimulus vector. Each time the artificial filter is inserted a spike is recorded immediately proceeding it. At this point we now have a stimulus vector and a corresponding spike vector, and the stimulus that precedes each spike is exactly the same. If the implementation of finding the filter is correct it should directly recover the artificial filter.

In practice the recorded spikes are not necessarily spaced far apart. It happens on occasion that the stimuli corresponding to two different spikes overlap each other. In order to capture this possibility the plots below show the recovered filter when 20 filters are inserted (no overlap) and when 3000 filters are inserted (substantial overlap). The STA program exactly recovers the filter for 20 spikes, and recovers a similar filter to the original for 3000 spikes. Note that the data that

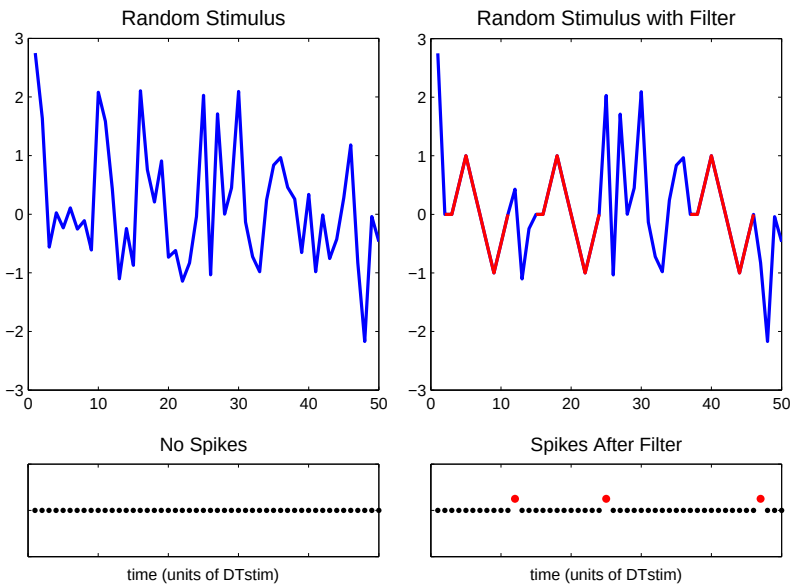

Figure 3: Visual representation of the creation of an artificial stimulus vector and corresponding spike vector for validation of the STA code.

we are working with has 14391 time steps in the stimulus filter and 2853 corresponding spike times.

The validation of the histogram method is not as straightforward as it is for the filter. For an explanation and proof of the validation please see appendix A.

3.2 The Spike-Triggered Covariance

The Spike-Triggered Covariance (STC), much like the STA, uses the idea of projection onto linear subspaces of the stimulus to reduce the dimensionality of the input to the model while still allowing the reduced input to maintain the salient features of the original.

Interpreting the STA and the STC geometrically can be useful in visualizing the process [4]. There are many points in stimulus space, the dimension of which is determined by the filter size we choose. Some of these points represent stimuli that did not trigger a spike (the gray points in figure 5(a)) and others represent stimuli that *did* trigger a spike (the black dots in figure 5(a)). The STA can be interpreted as the difference between the means of the raw stimulus data (black and gray points) and the spike-triggering stimulus data (the black points).

The STC builds on this idea and is defined as the difference between the variances of the raw stimulus data and the spike-triggering stimulus data. In practice the easiest way to interpret the data is by projecting out the STA from the stimulus first, and then finding the covariance of the spike-triggering stimuli:

$$STC = \frac{1}{N-1}\sum_{n=1}^{N}\left[\mathbf{s}(t_n) - \frac{STA\cdot\mathbf{s}(t_n)}{STA\cdot STA}STA\right]\left[\mathbf{s}(t_n) - \frac{STA\cdot\mathbf{s}(t_n)}{STA\cdot STA}STA\right]^T \tag{3}$$

This is shown in figure 5(b), where the STA (equation 2) has been projected out of the raw stimulus data.

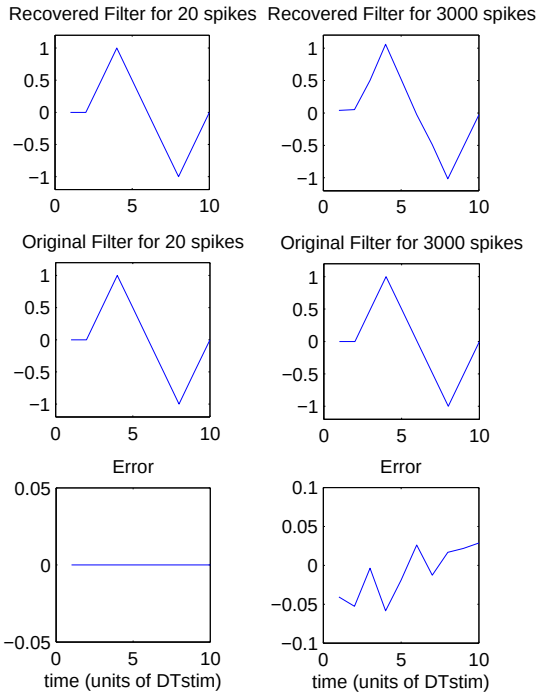

Figure 4: Top: Plot of the filter recovered by the STA program. Middle: Plot of the artificial filter created to validate the STA program. Bottom: Error between artificial filter and recovered filter. Artificial filter is 10 time steps long, with no upsampling.

Once we have constructed the STC from the data, we want to perform what is essentially a principal component analysis on the STC to ascertain which directions in stimulus space have the smallest and largest variances. The principal component analysis will be carried out on the matrix

$$\Gamma = \frac{1}{N-1}\sum_{n=1}^{N}\left[\mathbf{s}(t_n) - \frac{STA\cdot\mathbf{s}(t_n)}{STA\cdot STA}STA\right]\left[\mathbf{s}(t_n) - \frac{STA\cdot\mathbf{s}(t_n)}{STA\cdot STA}STA\right]^T - \mathbf{s}(t_n)\mathbf{s}(t_n)^T \tag{4}$$

where the first term is the spike-triggered covariance, the second term is the covariance of the stimulus, and again the STA is defined in equation 2. Subtracting out this covariance will get rid of any correlations that are present in the stimulus.

For the purpose of this project we will only be interested in the direction with the smallest variance, though the technique is not limited to this. The direction of smallest variance is the eigenvector associated with the smallest eigenvalue. Any stimulus vector with a significant component along the direction of this eigenvector has a much lower chance of inducing a spike response, hence this direction is associated with an *inhibitory* neural response.

With this information in hand we can now use the STA, associated with an *excitatory* neural response, and this new vector recovered from the STC analysis, to construct a model that uses both of these subspace features to filter the data. The new model becomes

$$r(t) = F(\mathbf{k}_e\cdot\mathbf{s}(t), \mathbf{k}_i\cdot\mathbf{s}(t)) \tag{5}$$

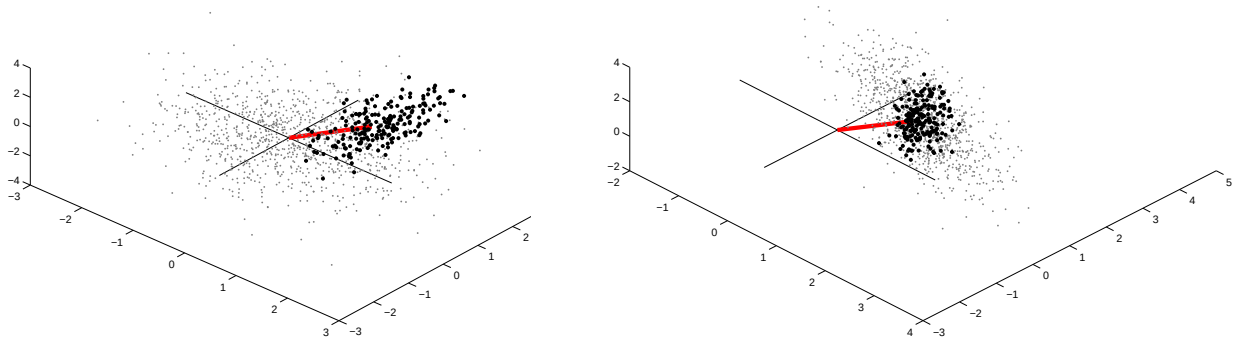

Figure 5: 3-dimensional representation of stimulus space helps to understand the STC. The gray points are stimuli that did not cause the neuron to spike, while black points are stimuli that did cause the neuron to spike. The red arrow is the STA, or the average of the black points. Right: The cloud of gray points is normally distributed about the origin. The cloud of black points has a different mean than the gray, which leads to the STA. Left: After projecting out the STA from the gray and black points, it is clear to see that there is a difference in the variance of the two clouds, which leads to the STC.

where $\mathbf{k}_e$ and $\mathbf{k}_i$ denote the excitatory and inhibitory filters, respectively. Notice that in general the STC method can be used to estimate parameters not only for the Linear-Nonlinear-Poisson model but for the more general nonlinear model shown in equation (5); $\mathbf{k}_e \cdot \mathbf{s}(t)$ and $\mathbf{k}_i \cdot \mathbf{s}(t)$ can be combined in any manner.

Now all that remains is to fit the nonlinear function F. Again we could fit a parametric form to the function and estimate its parameters, but like the STA technique we will use the histogram method, binning the possible values of $(\mathbf{k}_e \cdot \mathbf{s}(t), \mathbf{k}_i \cdot \mathbf{s}(t))$ and computing the average spike count for each bin. Notice that this method will work with at most two filters (visually, at least); with more than two filters parameter estimation would be a better choice.

3.2.1 STC Implementation

The STC algorithm builds on the STA algorithm, so much of the work is already done. Once the STA has been calculated the next step is to project out the STA from each point in stimulus space. If the STA has length s, then we want to look at a window of length s that slides along the stimulus vector one time step at a time, projecting out the STA at each step. Once this is done we can find the covariance matrix for the entire stimulus. This step is not important if the stimulus is uncorrelated, which should be the case; however, upsampling the stimulus to get a better time resolution introduces correlations in the stimulus, which will be important to keep track of for later.

At the same time we project the STA out of the stimulus vector, we can keep track of the stimuli that caused a spike and calculate the covariance matrix for this set as well. In terms of figure 5(b), the covariance of all stimuli will tell us about the variance of the cloud of gray points and the covariance of the spike-triggering stimuli will tell us about the variance of the cloud of black points.

By subtracting the stimulus covariance from the spike-triggering covariance we subtract out the effects of the potentially correlated stimuli.

The next step is to find the eigenvectors and eigenvalues of the resulting covariance matrix.

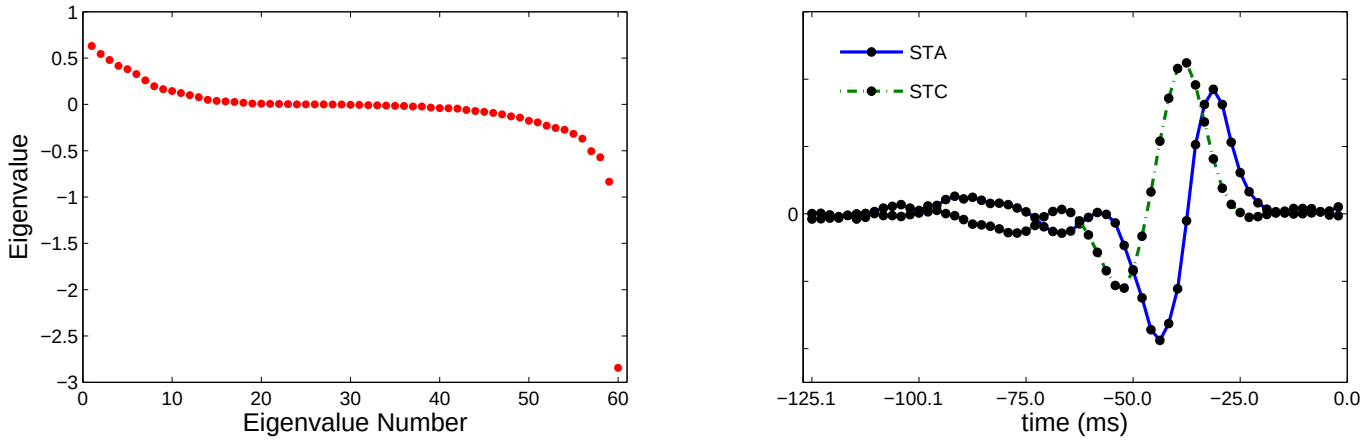

Figure 6: Left: Eigenvalues of the shifted spike-triggered covariance matrix. The large negative eigenvalue has an eigenvector that corresponds to an inhibitory filter. Right: STA and STC filters recovered from the STC algorithm. Plot is for a filter length of 15 times steps and an upsampling factor of 4.

The eigenvectors correspond to directions in stimulus space where the spike-triggering stimuli have a larger variance than the entire stimulus (positive eigenvalues), no difference in variance (zero eigenvalues), or a smaller variance (negative eigenvalues). We are interested in the eigenvalue corresponding to the smallest eigenvector (the negative eigenvalue with the largest magnitude), which gives us an inhibitory filter.

Figure 6(a) shows the eigenvalues of this shifted covariance matrix for a filter of length 15 and an upsampling factor of 4. Notice most of the eigenvalues cluster around zero, except for one eigenvalue that appears much smaller than the rest. In reference [4], the authors employ statistics to address how many of the eigenvalues are significantly different than zero to capture the most information possible in the model, but again we will only concern ourselves with the smallest eigenvalue for this project.

Figure 6(b) shows the resulting STA (excitatory) and STC (inhibitory) filters. Physically speaking, any stimulus sequence that looks like the STC will decrease the probability of the neuron spiking, whereas any stimulus sequence that looks like the STA will increase the probability of the neuron spiking.

After the STA and STC have been recovered from the data, the next step is to determine the neuron's response function, which will again be found using the "histogram method". Like the estimation of the filters, most of the work has already been done for the STA. All that is required now is to extend the algorithm to two dimensions. Figure 7 shows the plot of this discretized version of the response function, along with information detailing the distributions of the generating signals and the spikes counts along the direction of the STA and the STC.

Notice that, even though the resolution of the discretized function is low, it is possible to see the effects of the inhibitory filter. Large values of the generator signal in the direction of the STA (bottom of figure 7) have a high probability of causing a spike. However, the probability is de-

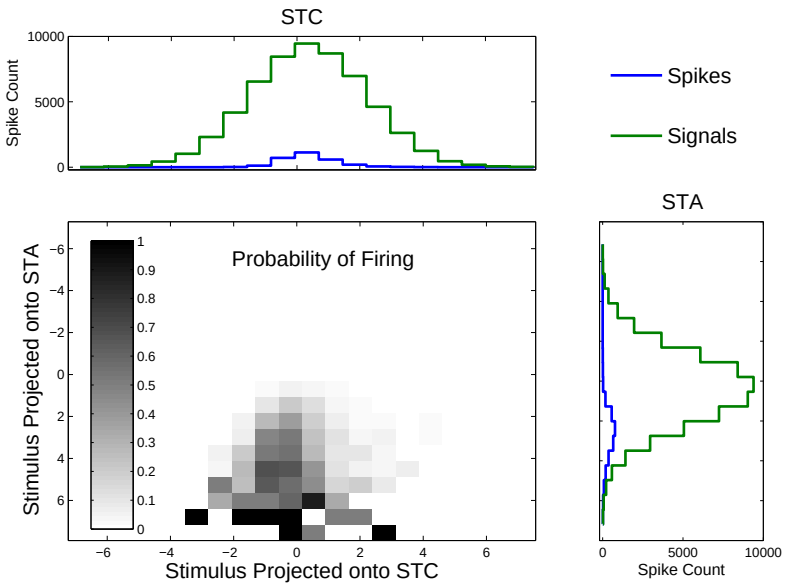

Figure 7: Discretized version of the nonlinear response function. Darker bins represent a higher average spike count. The plots on the side give the histogram of the generator signal count (the stimulus dotted with the respective filter) and the histogram of spike counts. The average number of spikes per bin in each of these directions is the spike count divided by the generator signal count for each bin.

creased for larger values of the generator signal in the direction of the STC (right of figure 7).

One failing of the histogram method in two (or more dimensions) is the poor scaling property. If we want to increase the number of bins in each direction by a factor of n, the total number of bins increases by n^2. Since there is only a finite amount of data, this scaling will spread out the spikes and leave the nonlinear response function with many bins that contain no spikes, as can already be seen in the figure.

3.2.2 STC Validation

The implementation of the STC is relatively straightforward, so there is little in the way of validation. For the part of the algorithm that actually finds the direction of the STC in stimulus space, recall that we are finding the covariance matrix of the spike-triggering stimuli and the covariance matrix of all the stimuli, and taking their difference. We then find the eigenvectors and eigenvalues of this difference.

If the code is working properly, and we then change every stimulus to be a spike-triggering stimulus, the difference should be zero and the resulting eigenvalues will all be zero. Furthermore, the associated eigenvectors will all be orthonormal vectors that span the stimulus space, which is just the standard basis. The results of this procedure are shown in figure 8(a). All eigenvalues are zero, and the recovered STC (which is the eigenvector that corresponds to the smallest eigenvalue) is simply the first vector in the standard basis, as expected.

The second part of the STC algorithm involves finding the discretized function of the spiking probability given the STA and the STC. This is a simple generalization of the code used in the STA

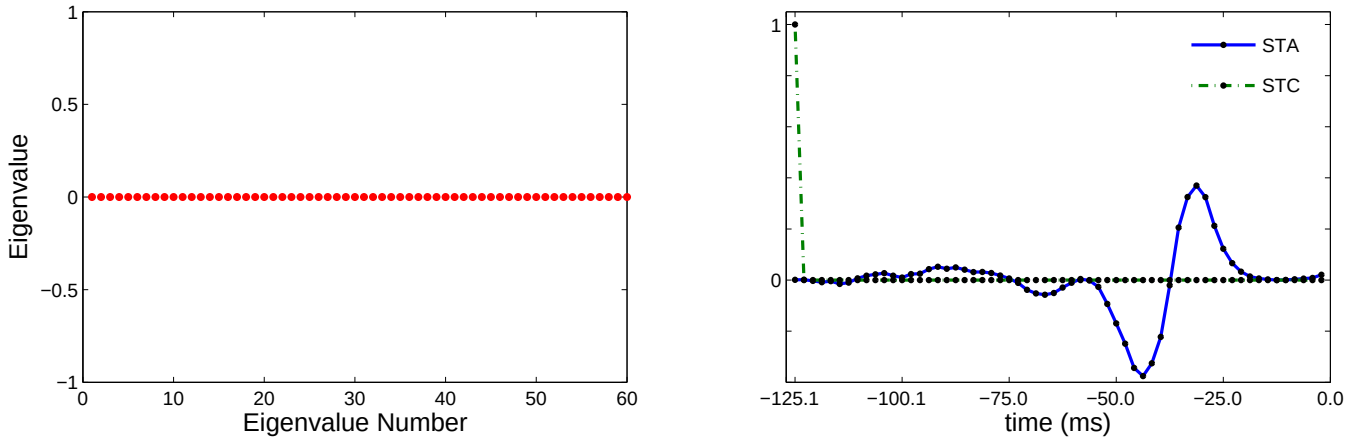

Figure 8: Validation of the STC algorithm

section to two dimensions, and as such I did not validate the implementation. Please see appendix A for more information.

4 Maximum Likelihood Estimators

The STA and STC algorithms are attractive in their simplicity, but are not powerful enough to create more general models. One significant drawback is that the parameters of the STA (and STC) can only be fit using spherically symmetric (normally distributed) stimuli [4]. The moment based techniques also do not allow us to include other predictors in the model, such as spike history dependence or network effects. For these reasons we move on to models that employ a much more general technique for parameter estimation, maximum likelihood estimates.

4.1 The Generalized Linear Model

The Generalized Linear Model (GLM) is one such way to accomplish this task of improving parameter estimation by maximum likelihood estimates. We will take a step back into the realm of linear models to explain the GLM and the maximum likelihood estimates, then apply these techniques to nonlinear models in later sections.

The LNP model of neural response produces a rate parameter $r(t)$ to an inhomogeneous Poisson process. If n is the number of observed spikes in a short time Δt, the conditional probability of this event is given by

$$P(n|r(t)) = \frac{(r(t)\Delta t)^n}{n!} e^{-r(t)\Delta t}. \tag{6}$$

If we are now interested in observing an entire spike train $\{n_t\}_{t=1}^{T}$, which is a vector of (assumed independent) spike counts binned at a time resolution of Δt, the conditional probability of this

event is given by

$$P(\{n_t\}|\{r(t)\}) = \prod_{t=1}^{T} \frac{(r(t)\Delta t)^{n_t}}{n_t!} e^{-r(t)\Delta t} \tag{7}$$

where the product runs over each time bin and $\{r(t)\}$ is the collection of rates, one for each element in $\{n_t\}$. Normally we would view this equation as the probability of the event $\{n_t\}$ given the collection of rate parameters $\{r(t)\}$. Instead we want to look at it from a different perspective: what is the likelihood that the collection of rate parameters is $\{r(t)\}$, given that the outcome was $\{n_t\}$ (and that we are using a Poisson distribution)? Viewed in this way equation (7) becomes a function of the collection of rate parameters $\{r(t)\}$, and is known as the *likelihood* function [7], which we will denote with an L:

$$L(\{n_t\}|\{r(t)\}) = \prod_{t=1}^{T} \frac{(r(t)\Delta t)^{n_t}}{n_t!} e^{-r(t)\Delta t}. \tag{8}$$

The maximum value of this function, known as the maximum likelihood, will be located at the values of the parameters of equation (1) that are most likely to produce the spike train $\{n_t\}$, and these are the parameters that we wish to find.

In practice it is easier to work with the log-likelihood function, since it transforms the product into a sum. The parameters that maximize the log-likelihood function will be the same parameters that maximize the likelihood function due to the monotonicity of the log function. The log-likelihood is often denoted using $\mathcal{L}$ so that, after taking the log of the likelihood function and ignoring constant terms, equation (8) becomes

$$\mathcal{L}(\{n_t\}|\{r(t)\}) = \sum_{t=1}^{T} n_t \log(r(t)\Delta t) - \sum_{t=1}^{T} r(t)\Delta t. \tag{9}$$

This is where we begin to develop a model of this process. In practice we do not know what the rates $\{r(t)\}$ are for a given neuron, and so our goal will be to model them using some parametric form of the rate $r(t)$. The following sections will detail just how we can model $r(t)$, but for now the important point to keep in mind is that these parametric models will be defined by a set of parameters that we will denote θ, as well as some sort of data that we will need to fit those parameters, which we will denote D. Then the likelihood in equation 9 becomes a likelihood of the parameters θ, so that

$$\mathcal{L}(\{n_t\}|\theta, D) = \sum_{t=1}^{T} n_t \log(r(t;\theta, D)\Delta t) - \sum_{t=1}^{T} r(t;\theta, D)\Delta t, \tag{10}$$

where $r(t;\theta, D)$ indicates that $r(t)$ depends on the parameters θ and the data D.

At this point we have an optimization problem to solve involving the parameters θ (which will encompass the linear filter $\mathbf{k}$ and the nonlinear function F). Fortunately, it has been shown by Paninski in [8] that with two reasonable restrictions on the nonlinear function F the log-likelihood function is guaranteed to have no non-local maxima, which avoids computational issues associated with numerical ascent techniques. The restrictions are 1) $F(u)$ is convex in its scalar argument u and 2) $log(F(u))$ is concave in u.

In the literature ([8],[12],[16]) it is common to choose a parametric form of F that follows these

restrictions, like $F(u) = e^u$ or $F(u) = \log(1 + e^u)$, and then optimize the function over the filter $\mathbf{k}$ and the parameters of the function F. The use of maximum likelihood estimates for the parameters of the model is a powerful technique that extends the nonlinear models considered later in the paper.

4.1.1 GLM Implementation

The difficulty in implementing the GLM is coding the log-likelihood function $\mathcal{L}$ in an efficient manner, since it is going to be evaluated numerous times by the optimization routine. Along with the function itself, the gradient needs to be evaluated at every iteration, adding additional time requirements.

To simplify equation 9, instead of thinking about the rate parameter $r(t)$ as given in Hertz (spikes per second), we can consider it to be the rate of spikes per bin. With this interpretation of the rate, $r(t)\Delta t$ then just becomes a scaled rate r_{bin}.

Once the log-likelihood function has been coded the GLM implementation simply reduces to an unconstrained optimization problem. To solve this optimization problem I began by implementing the gradient descent method, which proved to work but took too much time to be practical. I then decided to implement a Newton-Raphson method in the hopes of speeding up the optimization time. While the number of function evaluations dropped, the time for each function evaluation increased due to the need for the Hessian update at every iteration. Next I implemented a quasi-Newton method, which is a nice compromise between gradient descent and Newton-Raphson: it doesn't involve the cost of computing the Hessian, and converges with fewer function evaluations than the gradient descent.

I implemented the Broyden-Fletcher-Goldfarb-Shanno (BFGS) variant of the quasi-Newton method, and validated it on a few test cases found online. I also found that my implementation was able to optimize the log likelihood function employed in the GLM. For a full description of my implementation of the BFGS routine, please see appendix B.

As mentioned above there is a particular form for F that we can use to guarantee a non-local minimum. The functional form that I have chosen to use is $F(u) = log(1+exp(u))$. Using this form the complete log-likelihood function (and objective function of the optimization problem) becomes

$$\mathcal{L}(\{r_{bin}(t)\}|\{n_t\}) = \sum_{t=1}^{T} n_t \log(\log(1 + e^{\mathbf{k}\cdot\mathbf{s}(t)+a})) - \sum_{t=1}^{T} \log(1 + e^{\mathbf{k}\cdot\mathbf{s}(t)+a}). \tag{11}$$

The optimization routine will find the (global) maximum log-likelihood, which will be at particular values for $\mathbf{k}$ and a. a is an offset parameter that establishes the background firing rate of the model in the absence of stimuli. As in the STA method, upsampling the stimulus vector results in an increased resolution of the filter.

The optimization routine simultaneously finds the nonlinear function parameters; in this case the function offset a. Like the STA, it is more instructive to view the resulting function relative to the distribution of the generator signal, as shown in figure 9(b).

4.1.2 GLM Regularization

The next step in implementing the GLM is to introduce a regularization term. Regularization helps the model to avoid overfitting, and also allows us to introduce a priori knowledge of the

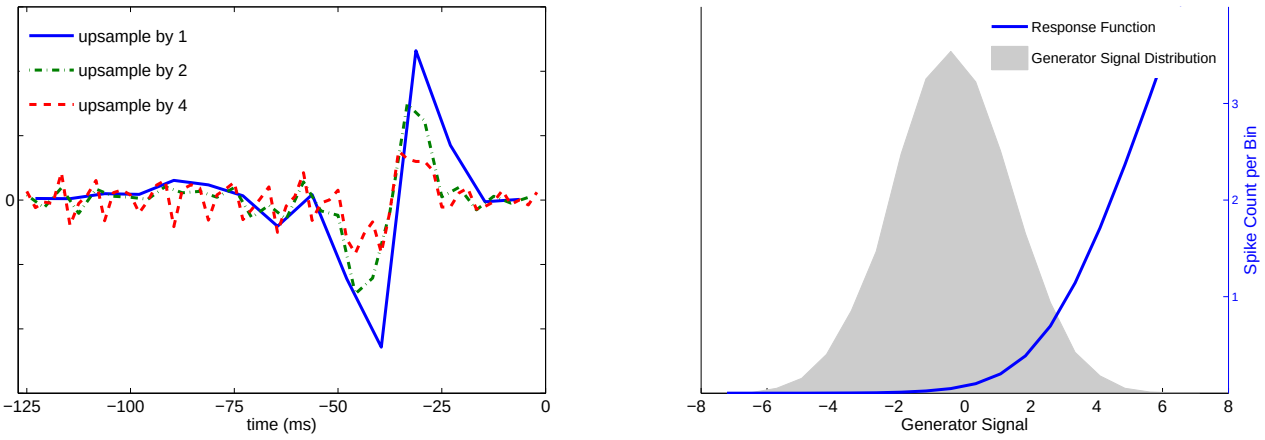

Figure 9: Left: Filters found using the GLM for various upsampling factors. Right: Response function for a filter of length 15 time steps and an upsampling factor of 1, along with the histogram of generator signals.

solution. For a stimulus that is one-dimensional in time, avoiding overfitting amounts to penalizing the curvature of the resulting filter; large curvatures indicate large fluctuations, which is typical in overfitting. To reduce the total curvature of the filter, we can add a term to the log-likelihood function that penalizes large values of the second derivative of the filter, which is given by the L^2 norm of the discrete Laplacian applied to the filter. The discrete Laplacian is denoted as L^t and is defined as a matrix that acts on a vector (in this case the vector is the filter) and returns the difference between a component of the vector a_i and the average of it's two neighbors a_{i-1} and a_{i+1}. The edge elements are computed by linearly extrapolating the Laplacian values from the interior points, so the matrix becomes

$$L^t = \frac{1}{2}\begin{pmatrix} 2 & -5 & 4 & -1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & -2 & 1 & 0 & \cdots & 0 \\ \vdots & & \ddots & \ddots & \ddots & & 0 \\ \vdots & & & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 0 & 1 & -2 & 1 \\ 0 & \cdots & 0 & -1 & 4 & -5 & 2 \end{pmatrix}$$

The log-likelihood function then becomes

$$\mathcal{L}(\{r_{bin}(t)\}|\{n_t\}) = \sum_{t=1}^{T} n_t \log(r_{bin}(t)) - \sum_{t=1}^{T} r_{bin}(t) - \lambda \|L^t \mathbf{k}\|_2^2, \tag{12}$$

where L^t is the discrete Laplacian in the time dimension, and λ modulates the effect of the regularization term.

Now the question is, how do we choose an appropriate value of λ? It is not technically a parameter of the model, but still a parameter that needs to be optimized, and is hence called a *hyperparameter*. To optimize λ I employ cross validation.

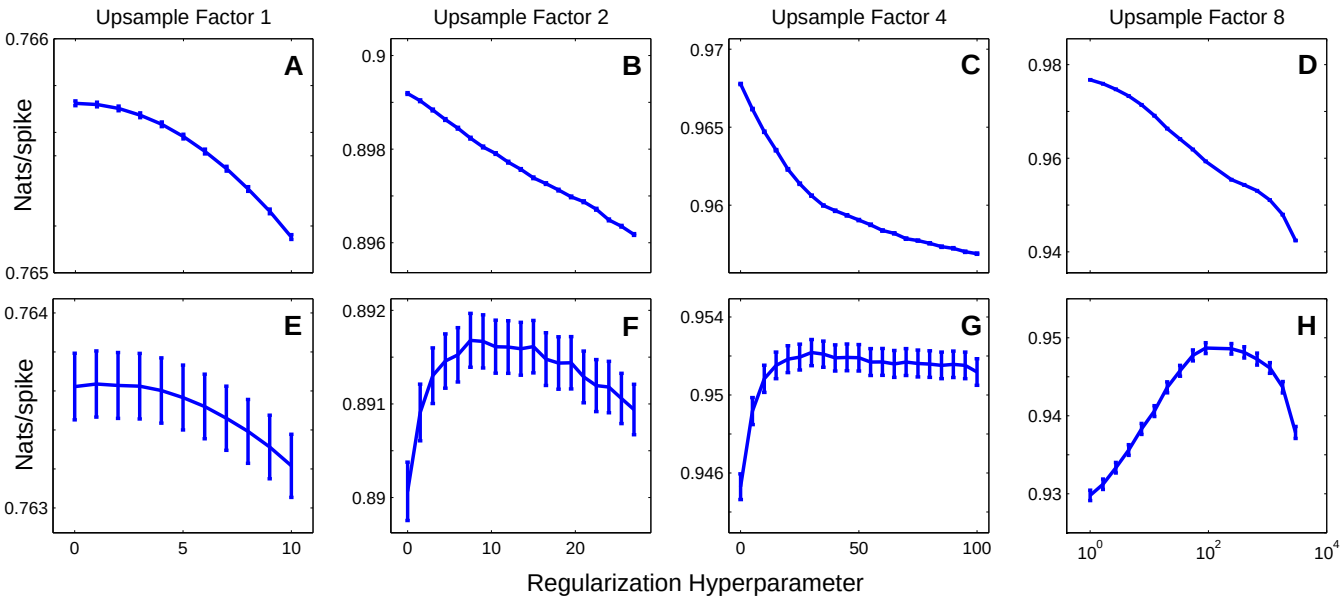

Figure 10: Results from the cross validation of the regularization hyperparameter. The top row (plots A-D) shows the number of nats/spike for the test data. The bottom row (plots E-H) shows the corresponding values of nats/spike for the validation data. The maximum nats/spike for the validation data is 0.9522 for an upsampling factor of 4, and 0.9487 for an upsampling factor of 8. All plots are for a filter length of 15 time steps.

The first step is to divide the data into a number of equally sized pieces, called folds; in the plots that follow, the data is divided into five folds. I choose an arbitrary value for λ and fit the model parameters using four of the folds (the test data). I then use the remaining fold (the validation data) to determine how well the model generalizes to novel data by evaluating the likelihood function with the validation data. This gives a value of the likelihood function for the particular value of λ used.

This process is repeated using each of the five folds for the validation data, which will give five values of the likelihood for the particular value of λ chosen. The value of the likelihood for this one value of λ can then be computed by averaging the five values, or by more sophisticated methods. This process is now repeated for different values of λ and the value that produces the largest value of the likelihood is the optimal value of λ. Put another way, this is the value that maximizes the likelihood that the given model produced the observed spike vector.

The plots go a step further than the process described above. For a single value of λ there are five folds, which lead to five values of the likelihood. One way to account for variances that might arise from random starting positions in the optimization routine is to average over many starting positions. For each fold the likelihood is averaged over ten random starting positions, so that there are fifty likelihood values. The average of these fifty values is plotted, along with error bars indicating the standard deviation of the fifty values.

There is a difficulty in comparing likelihood values across models using different upsampling factors, because the likelihood will depend on the bin size used. In order to properly compare these models we look at the relative log likelihood (RLL) of each model, which is associated with the

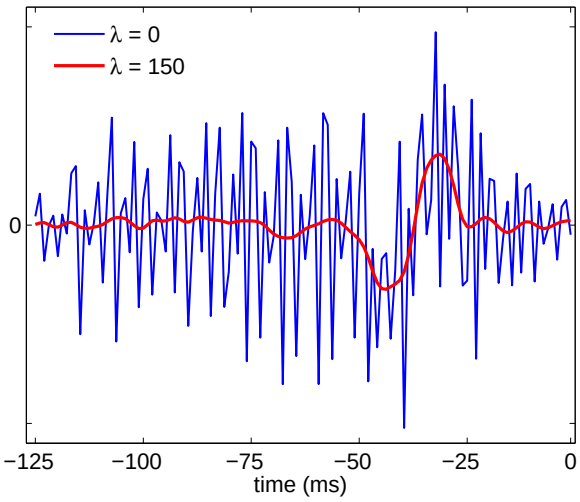

Figure 11: Resulting filters after regularization term has been included for an upsampling factor of 8, showing its effect on the smoothness of the filter. The optimal value at $\lambda = 150$ is shown along with the unregularized filter.

single spike information of the neuron in question. For a complete explanation of the relative log likelihood measure, please see appendix C.

Figures 10 A-D show the value of the RLL using the test data that the model was fitted with, for upsampling factors of 1, 2, 4 and 8 respectively. Figures 10 E-H, on the other hand, show the RLL for the validation data.

The plots for an upsampling value of 8 show clear evidence of overfitting. The RLL for the test data starts at a maximum value, indicating a good fit, and decreases as the regularization parameter increases, indicating an increasingly worse fit. However, the RLL for the model on the validation data *increases* during this same interval, indicating the model was initially overfitting the data but is now able to generalize better. At $\lambda = e^5 \approx 150$, the RLL of the validation data reaches a maximum. Figure 11 shows the filter with hyperparameters of $\lambda = 0$ (no regularization) and $\lambda = 150$ (optimal regularization).

The plots for an upsampling value of 1 don't show the same behavior. The RLL of the test data does decrease with increasing regularization, but the validation data decreases as well. This implies that regularization is not needed when there is no upsampling; there are not enough parameters to allow a significant degree of overfitting. The values of the RLL in figure 10 show a marked increase as the upsampling factor is increased from 1 to 2 and from 2 to 4, but decreases as the upsampling factor goes from 4 to 8. Figure 12 plots the dependence of the RLL on the upsampling factor, and shows that it reaches a maximum around 6. The reasons for this have to do with the time scales of the stimulus used, and are particular to the data set. The details are discussed further in reference [15].

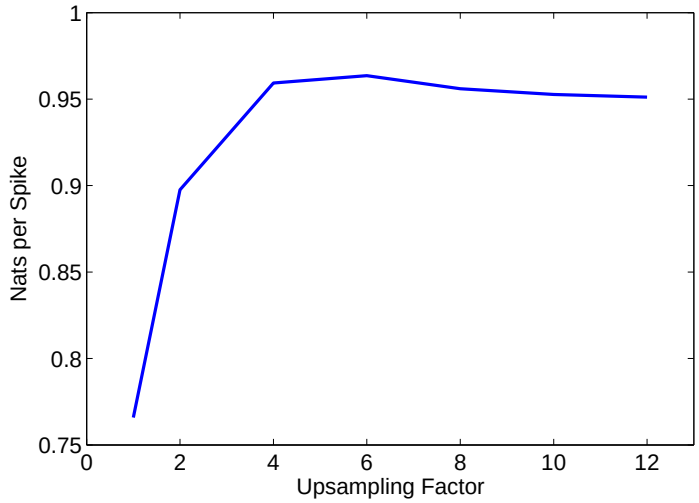

Figure 12: Value of the single spike information as the upsampling factor is varied.

4.1.3 GLM Validation

To validate the GLM model I employ the fact that Maximum Likelihood Estimates are consistent, which means that for large enough sample sizes the estimates can be arbitrarily close to the true values. More precisely, if $\hat{\theta}_n$ represents all of the parameters for the GLM model, estimated using the maximum likelihood for a sample size of n, and if θ_0 represents the true parameters, then a consistent estimator is one such that, for any $\epsilon > 0$, ([11])

$$\lim_{n\to\infty} P(|\hat{\theta}_n - \theta_0| \geq \epsilon) = 0$$

To use this for validation purposes we can create an artificial filter, and from that filter we can use the GLM to create firing rates. From these firing rates we can then reproduce spike trains using the probability of firing in each bin. With these spikes trains and a stimulus of Gaussian random noise we run the GLM program and hope to recover the original filter that the data was created with. Due to the consistency of the Maximum Likelihood Estimates, increasing the length of the spike train should decrease the error between the estimated filter and the filter we used to create the data.

Figure 13(b) shows the mean square error (MSE) between the generating filter and the estimated filter as a function of sample size. The MSE for the estimates do decrease with increasing sample size, though it seems to level out at a nonzero value. Even still, the filter visually looks very close the generating filter (figure 13(a), and my computer does not have enough memory to create larger datasets in which to test the asymptotic values of the MSE.

4.2 Nonlinear Models

What makes the linear models like the GLM attractive is their ability to fit the data well, the tractability in estimating their parameters, and the fact that some of these parameters can be interpreted biologically. However, this method of linear stimulus processing fails to capture some of the more subtle features of a neuron's response; this is where the nonlinear models come in.

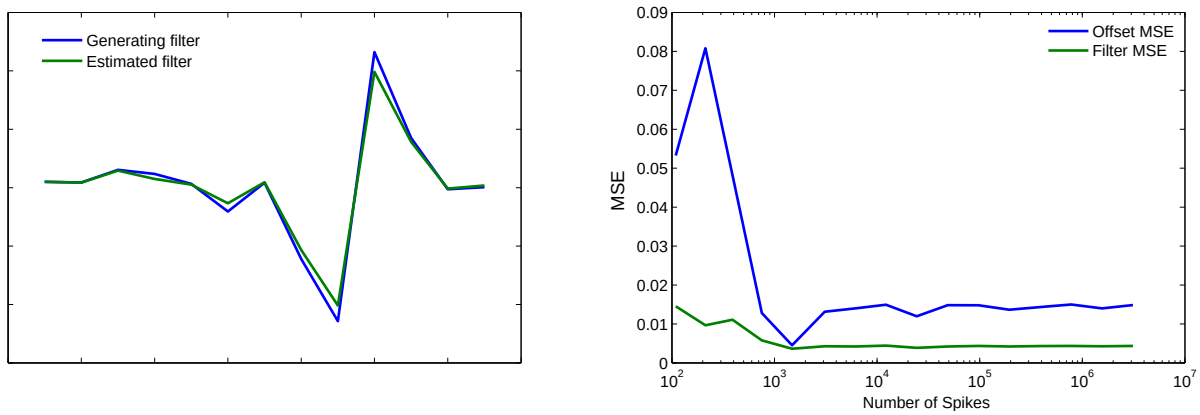

Figure 13: Left: Filter used to generate the spike train, and the estimated filter using the GLM for 10000 time bins. There were approximately 1800 spikes. Right: The mean square error between the generating parameters and the estimated parameters for increasing sample size, measured by the number of spikes present.

The nonlinear models are a natural extension of the linear model, and in their general form are given by the equation

$$r(t) = F\big(f_1(\mathbf{k_1} \cdot \mathbf{s}(t)), f_2(\mathbf{k_2} \cdot \mathbf{s}(t)), \ldots, f_n(\mathbf{k_n} \cdot \mathbf{s}(t))\big) \tag{13}$$

where the f_i's can be combined in any manner. Increasing evidence in neuroscience literature suggests that neural processing is performed by summing over excitatory and inhibitory inputs [16]; this fact, combined with increased ease of parameter estimation, leads us to make the assumption that the inputs of the nonlinear models will be combined as a weighted sum, in which case the nonlinear models are given by the equation

$$r(t) = F\bigg(\sum_i f_i(\mathbf{k_i} \cdot \mathbf{s}(t))\bigg). \tag{14}$$

The next two sections will examine particular instances of equation (14) and how the parameters are fitted.

4.3 The Generalized Quadratic Model

The Generalized Quadratic Model (GQM) is an intuitive first step into the realm of nonlinear models. It simply adds a quadratic term and a constant term to the LNP model considered in equation (1), given by

$$r(t) = F\bigg(\frac{1}{2}\mathbf{s}(t)C\mathbf{s}(t) + \mathbf{b}^T\mathbf{s}(t) + a\bigg), \tag{15}$$

where C is a symmetric matrix, $\mathbf{b}$ is a vector, and a is a scalar [12]. Similar to the implementation of the GLM, we want to choose a parametric form for the nonlinearity F and maximize the resulting log-likelihood function to estimate the parameter values of C, $\mathbf{b}$ and a.

4.3.1 GQM Implementation

There are two immediate problems in transitioning from the GLM to the GQM if we want to continue using maximum likelihood estimates. The first is that the argument of the spiking nonlinearity F is no longer convex, and the guarantee of a single global maximum for the log-likelihood function given in [8] does not apply. However, in practice this function is still well behaved ([16]) and the same optimization procedures can be used.

The second problem is that with the introduction of the matrix C, the number of parameters to fit becomes much larger than before. With a filter size of 15 times steps, the GLM has 16 parameters to fit (with one extra for the constant in the spiking nonlinearity). Now with a filter size of 15 steps, we have $15^2 + 15 + 1 = 241$ parameters to fit. In practice we can restrict C to be symmetric, but this still leaves us with a much larger optimization problem.

Several approaches have been proposed to address this problem ([12], [13]), both of which attempt to reduce the number of parameters to estimate. One way to do this is to make a low-rank approximation to C, so that

$$C = \sum_{n=1}^{N} w_n \mathbf{k}_n \mathbf{k}_n^T, \tag{16}$$

which will have rank N. Each component of C is weighted with a constant w_n, which is restricted to be $\{\pm 1\}$. Choosing the rank N and the weights w_n will be addressed in the following section. In this manner a filter can be interpreted as an excitatory input ($w_n = 1$) or an inhibitory input ($w_n = -1$) and the GQM becomes

$$r(t) = F\left(\frac{1}{2}\sum_{n=1}^{N}(\mathbf{k}_n \cdot \mathbf{s}(t))^2 + \mathbf{b}^T\mathbf{s}(t) + a\right). \tag{17}$$

4.3.2 GQM Model Selection

The question of how to choose the rank N of the matrix C is a question of model selection, and we are faced with certain trade-offs: we can find an increasingly good fit with lots of parameters, but we also want to limit the degree of model complexity for ease of estimation, among other reasons.

Before we address this question, let's redefine the problem in a way that will be more useful. Instead of choosing the rank N of C, let us say there are N_+ exitatory filters and N_- inhibitory filters; in this case, equation (17) becomes

$$r(t) = F\left(\frac{1}{2}\sum_{e=1}^{N_+}(\mathbf{k_e} \cdot \mathbf{s}(t))^2 - \frac{1}{2}\sum_{i=1}^{N_-}(\mathbf{k_i} \cdot \mathbf{s}(t))^2 + \mathbf{b}^T\mathbf{s}(t) + a\right). \tag{18}$$

Now we want to choose two numbers, N_+ and N_-, that give the "best" model from this class of models (the GQM class). We will use two different metrics, Akaike's Information Criterion (AIC) and Bayesian Information Criterion (BIC). A comprehensive review of these two model selection criteria can be found in [14].

AIC uses ideas from information theory, specifically the Kullback-Leibler (K-L) Information (or Kullback-Leibler Divergence). We assume that there is a "true" probability distribution f, and we wish to approximate that probability distribution with another distribution g, in this case the one

given by our model. The K-L Information is a measure that tells us how much information we have lost in our approximation, and is given by

$$I(f,g) = \int f(x) \log\left(\frac{f(x)}{g(x|\theta)}\right) dx, \tag{19}$$

where θ is a set of parameters used to specify g. Ideally we would like to find the density g that minimizes the information lost. The above equation can be rewritten as

$$I(f,g) = \mathbb{E}_f[\log(f(x))] - \mathbb{E}_f[\log(g(x|\theta))], \tag{20}$$

where the expectations are taken over the probability distribution f. The first expectation in the expression is fixed for all possible models (since it represents the "truth"), and so we want to maximize the second expectation. Akaike was able to show that an asymptotically unbiased estimator of this expectation is

$$\mathcal{L}(\hat{\theta}|data) - K, \tag{21}$$

where $\mathcal{L}$ is the maximized log-likelihood function, $\hat{\theta}$ are the maximum likelihood estimates of the model parameters given the data, and K is the number of estimable parameters in the model. The AIC is then defined as

$$AIC = -2\mathcal{L}(\hat{\theta}|data) + 2K \tag{22}$$

and smaller values of the AIC represent less information loss.

To implement the AIC for the GQM model, the above equation that we would like to minimize becomes

$$AIC = -2\mathcal{L}(\hat{\theta}|data) + 2\big((N_+ + N_- + 1)s + 1\big), \tag{23}$$

where s is the length of each individual filter. We need to compute the AIC for various combinations of N_+ and N_- through an exhaustive search of this parameter space, where each number can range over the nonnegative integers. The procedure is as follows:

1. For a given pair of values (N_+, N_-), find the optimum regularization constant λ using k-fold cross validation.

2. Find the optimum parameters (filters) using the entire data set (which corresponds to $\hat{\theta}$ in the above equations), and using the value of λ found in step 1.

3. Compute the log-likelihood using the optimum filters and the entire data set to get $\mathcal{L}(\hat{\theta}|data)$, and compute the AIC. The log-likelihood value computed at this step is *not* the penalized log-likelihood value that includes the regularization term found in step 2.

4. Repeat for next (N_+, N_-) combination.

5. Choose the values of (N_+, N_-) that minimize the AIC.

In actually implementing this procedure we can repeat step 3 with 10 random initial starting conditions and average the log-likelihood over these 10 values for each (N_+, N_-) combination. Once all AIC values are computed we subtract off the smallest AIC value from each according to [14], since the absolute difference between the values is what matters when ranking the models. Then the "best" model has a value of 0, and the rest of the models have positive values that increase as the models get worse according to Akaike's Information Criterion.

The results are shown in the left in Tables 1 and 2; the parameter combination that results in the smallest AIC value has 3 inhibitory quadratic filters and 0 excitatory quadratic filters (the

$N_+ \backslash N_-$	0_-	1_-	2_-	3_-	4_-
0_+	1479.3	535.9	94.5	0	1.7
1_+	1483.1	540.4	100.4	6.9	7.4
2_+	1499.4	555.8	117.4	26.6	25.1
3_+	1518.7	579.3	140.7	49.7	56.8

$N_+ \backslash N_-$	0_-	1_-	2_-	3_-	4_-
0_+	1157.6	327.8	0	19.1	134.4
1_+	1275.0	445.9	119.58	139.7	253.8
2_+	1404.9	575.0	250.2	273.0	385.2
3_+	1537.9	712.1	387.1	409.6	530.4

Table 1: Right: AIC values for the GQM with an upsampling factor of 1. Left: BIC values for the GQM with an upsampling factor of 1. Columns show the number of inhibitory filters used in the model (denoted by N_-) and rows show the number of excitatory filters used in the model (denoted by N_+).

$N_+ \backslash N_-$	0_-	1_-	2_-	3_-	4_-
0_+	2641.0	553.4	61.7	0	43.5
1_+	2684.7	591.4	320.9	121.8	1023.3
2_+	2739.4	4788.7	-	-	-

$N_+ \backslash N_-$	0_-	1_-	2_-	3_-
0_+	2083.3	243.7	0	186.4
1_+	2375.0	529.7	507.2	556.2
2_+	2677.8	4975.0	-	-

Table 2: Right: AIC values for the GQM with an upsampling factor of 2. The smallest value corresponds to a regularization constant of 12. Left: BIC values for the GQM with an upsampling factor of 2. The smallest value corresponds to a regularization constant of 9. The regularization constant was allowed to take a value from 0 to 39 by increments of 3. Columns show the number of inhibitory filters used in the model (denoted by N_-) and rows show the number of excitatory filters used in the model (denoted by N_+).

linear filter b is an excitatory filter), which are all plotted in figure 14(a).

The second criterion we can use is the Bayesian Information Criterion, which was motivated by placing priors on the distributions rather than using information theory, though reference [14] shows that the BIC can be derived using methods similar to the AIC and vice-versa. The BIC takes a similar form to the AIC, but also takes into account the size of the data set being used to estimate the models. If the total number of data points used in model estimation is given by T, the BIC is

$$BIC = -2\mathcal{L}(\hat{\theta}|data) + \big((N_+ + N_- + 1)s + 1\big)\log(T). \tag{24}$$

The first thing to notice is that the BIC penalizes the number of parameters more heavily, since $\log(T)$ will be greater than 2 if we have 8 or more data points, which in this case we certainly do. The results of the BIC are shown in the right in Tables 1 and 2, and we can see that the BIC indeed penalizes parameters more heavily, and shows that 2 inhibitory quadratic filters and 0 excitatory quadratic filters is optimal. The debate about whether to use AIC or BIC has been long running in the literature, and the authors of [14] claim that the choice lies in a matter of philosophy and also depends on the nature of the experiment.

4.3.3 GQM Validation

To validate the implementation of the GQM we can again use the fact that the Maximum Likelihood Estimates are consistent estimators. We choose one inhibitory quadratic filter in addition to the linear filter and generate a spike train using the GQM model. Figure 15 shows the recovered quadratic filter, as well as the MSE as a function of sample size.

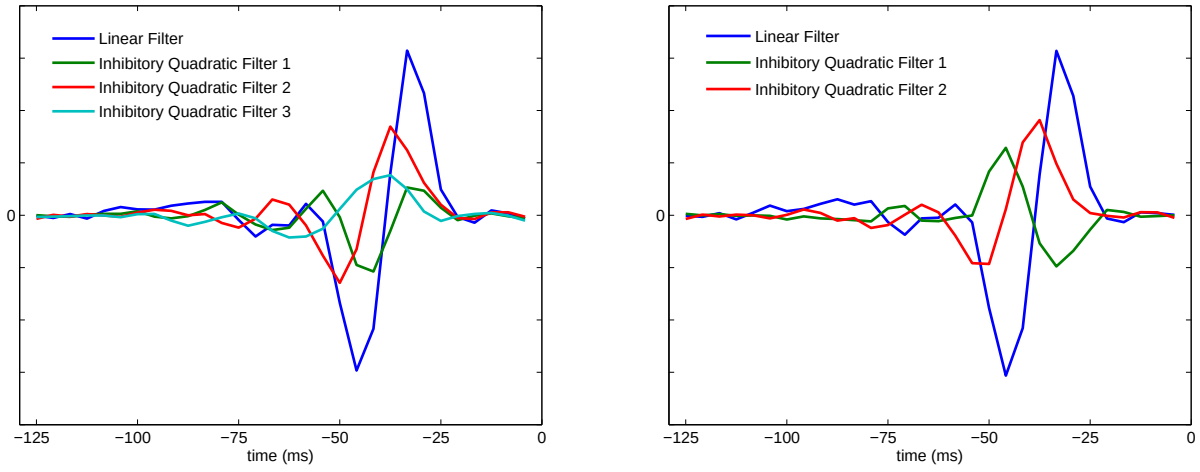

Figure 14: Left: GQM filters from minimum AIC value for an upsampling value of 2. Right: GQM filters from minimum BIC value for an upsampling factor of 2.

4.4 The Nonlinear Input Model

The implementation of the Nonlinear Input Model (NIM) is the overarching goal of this project. The NIM considers the Poisson rate parameter to be a function of a sum of nonlinear inputs that are weighted by ± 1, corresponding to excitatory or inhibitory inputs [16]. The equation, similar to equation (14), is given by

$$r(t) = F\left(\sum_i w_i f_i(\mathbf{k_i} \cdot \mathbf{s}(t))\right), \tag{25}$$

where the values of w_i are restricted to ± 1. This model can also be thought of as a two-layer LNP model, or an LNLN model: The stimulus $\mathbf{s}(t)$ is projected onto various subspaces by the filters k_i; the functions f_i transform these projections nonlinearly, and the results are summed linearly and used as an input to the larger nonlinear function F, which in turn gives a rate for the inhomogeneous Poisson process.

For the purposes of this project we will assume parametric forms of the f_i and F to make parameter fitting easier, though in practice the NIM can also fit these functions without an assumed parametric form using a set of basis functions. The f_i's will be rectified linear functions, where $f_i(u) = \max(0, u)$; F will be of the form $F = \log(1 + e^u)$, which guarantees no non-global maxima in the case of linear f_i's and will in practice be well-behaved for the rectified linear functions [16]. With these assumptions made, the gradient ascent routine will only need to optimize the filters k_i. Up to this point we have ignored the history dependence of a neuron when when trying to predict its output, and have instead focused solely on the stimulus dependence. For this implementation of the Nonlinear Input Model, we have included this history dependence. It is well established in neuroscience that neurons can display a variety of history-dependent behaviors. Because the very nature of an action potential involves changing the concentrations of ions inside the neuron, there is an absolute refractory period when the neuron cannot create another action while the ion concentrations reset. Past this point there is a relative refractory period, when the neuron is capable of spiking but only under a more intense stimulus. There are also behaviors like bursting,

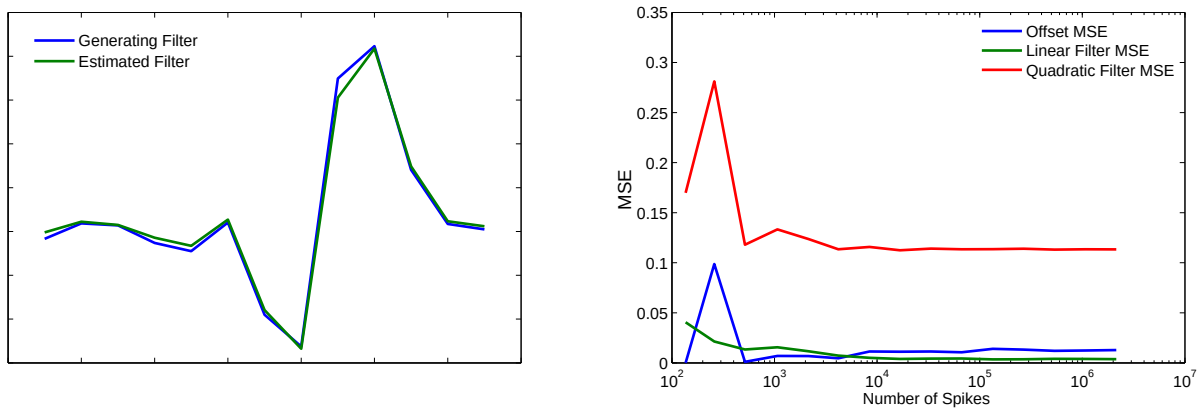

Figure 15: Left: Quadratic filter used to generate the spike train, and the estimated filter using the GQM for 10000 time bins. There were approximately 1800 spikes. Right: The mean square error between the generating parameters and the estimated parameters for increasing sample size, measured by the number of spikes present.

when a single action potential is followed by a series of action potentials in rapid succession [17]. It is clear that we should be interested in trying to characterize this history dependence as well.

Fortunately this addition to the model is relatively simple. We assume that the history dependence is linear, so that if $\mathbf{h}$ is a vector analogous to the filters $\mathbf{k}_i$ and $n(t)$ is a vector of the previous spikes counts over a certain temporal window at time t, then the NIM with this history component becomes

$$r(t) = F\left(\sum_i w_i f_i(\mathbf{k_i} \cdot \mathbf{s}(t)) + \mathbf{h} \cdot \mathbf{n}(t) + a \right).$$

The components of the history vector $\mathbf{h}$ may be large and positive for small time lags, which indicate that spikes very near time t will tend to increase the firing rate of the neuron; this would correspond to a bursting neuron. If the components are large and negative for small time lags, then spikes very near time t tend to decrease the firing rate and this is evidence of a refractory period (see figure 16(b)). I have completed the model selection and testing for the NIM with and without the history component, so that its effect can be seen on the model.

4.4.1 NIM Implementation

The implementation of the NIM is fairly straightforward once the GQM has been coded properly. All that remains is to change the nonlinear functions (and their associated gradients) from quadratic functions to rectified linear functions. The rectified linear functions are implemented in MATLAB by finding the linear functions, then using logical indexing to set all values less than 0 equal to 0.

For example, if A is a matrix of positive and negative values, and L is a matrix of 0s and 1s (a logical matrix), then the command $A(L) = 0$ will set every element of A equal to 0 where the matrix L has a 1. The derivatives are implemented in MATLAB by using the *step* function, which is 0 for any values less than 0 and 1 for any values greater than 0.

The added history component, because it is linear, may be optimized at the same time as the linear filters inside the nonlinearities f_i.

4.4.2 NIM Model Selection

Like the GQM, the equation for the NIM involves a sum over an undefined number of nonlinear functions, each associated with an excitatory or an inhibitory input. We will use the AIC and BIC as before to determine what the optimal number of excitatory and inhibitory inputs are. The results are given in Tables 3 and 4.

$N_+\backslash N_-$	1_-	2_-	3_-	4_-	5_-
0_+	3202.5	918.4	303.5	168.2	101.2
1_+	1301.7	311.5	79.5	0	44.3
2_+	1346.3	391.7	161.0	40.6	33.9

$N_+\backslash N_-$	0_-	1_-	2_-	3_-	4_-
0_+	5868.8	2782.1	611.7	110.4	88.7
1_+	1815.1	995.0	118.4	0	34.1
2_+	1943.0	1153.2	312.2	195.1	188.3

$N_+\backslash N_-$	3_-	4_-	5_-	6_-	7_-
0_+	768.7	275.1	145.1	64.7	39.0
1_+	339.7	83.5	28.9	0	18.16
2_+	379.8	170.6	66.4	19.6	17.1

$N_+\backslash N_-$	1_-	2_-	3_-	4_-	5_-
0_+	5063.4	1711.2	458.0	77.9	61.6
1_+	1509.2	593.6	142.6	0	59.0
2_+	1646.4	713.7	296.3	200.7	210.1

Table 3: Top Left (Right): AIC (BIC) values for the NIM with an upsampling factor of 1. Bottom Left (Right): AIC (BIC) values for the NIM with an upsampling factor of 1 when the history component has been added. Columns show the number of inhibitory filters used in the model and rows show the number of excitatory filters used in the model.

$N_+\backslash N_-$	2_-	3_-	4_-	5_-	6_-	7_-
0_+	1122.6	441.1	162.1	579.2	70.9	4050.3
1_+	292.0	11.5	71.5	75.9	0	42.0
2_+	315.7	229.8	6.5	81.8	132.8	160.4

$N_+\backslash N_-$	1_-	2_-	3_-	4_-
0_+	3851.7	615.0	181.5	150.6
1_+	1241.3	32.4	0	308.0
2_+	1537.9	304.2	466.3	491.0

$N_+\backslash N_-$	2_-	3_-	4_-	5_-	6_-	7_-
0_+	1247.6	526.4	247.6	165.8	88.0	0
1_+	568.4	274.6	103.9	72.0	64.7	63.13
2_+	590.7	358.3	193.6	128.2	167.84	69.6

$N_+\backslash N_-$	2_-	3_-	4_-	5_-
0_+	503.9	30.7	0	166.1
1_+	72.8	26.9	194.2	320.4
2_+	343.1	358.7	442.0	624.6

Table 4: Top Left (Right): AIC (BIC) values for the NIM with an upsampling factor of 2. The smallest value corresponds to a regularization constant of 15 (6). Bottom Left (Right): AIC (BIC) values for the NIM with an upsampling factor of 2 when the history component has been added. The smallest value corresponds to a regularization constant of 3 (3). The regularization constant was allowed to take a value from 0 to 21 using increments of 3, and is the same for all filters. Columns show the number of inhibitory filters used in the model and rows show the number of excitatory filters used in the model.

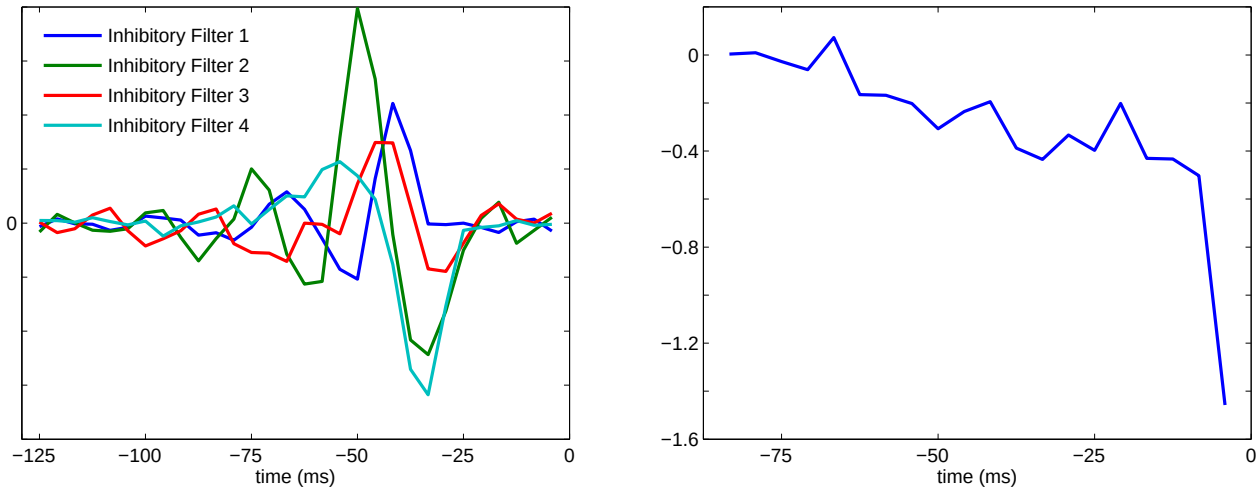

Figure 16: Left: NIM filters from minimum BIC value for an upsampling value of 2. Right: NIM history component for an upsampling factor of 2.

4.4.3 NIM Validation

To validate the implementation of the NIM we can again use the fact that the Maximum Likelihood Estimates are consistent estimators. We can choose one excitatory and one inhibitory nonlinear filter for the spike train generation, as well as a spike history term to mimic a refractory period. Figure 17(a) shows the recovered history term, for which the Maximum Likelihood Estimate was the least accurate, as seen in figure 17(b).

5 Databases & Implementation

The dataset that was used to develop the above models is from a Lateral Geniculate Nucleus neuron's response to a single pixel of temporally modulated white noise stimuli, found at http://www.clfs.umd.edu-/biology/ntlab/NIM/. The data consists of two parts, one for fitting and one for testing. The first is a stimulus vector that contains the pixel value, changing every 0.0083 seconds, for 120 seconds which gives a total of $14,391$ values (FFstim). Along with this is a vector that holds the time in seconds at which a spike was recorded (FFspks). The second part of the data consists of a 10 second stimulus (FFstimR), again changing every 0.0083 seconds, and 64 separate trials during which spike times were recorded (FFspksR).

6 Testing

Testing of the various models will be performed using two metrics: the cross-validated relative log likelihood and the fraction of variance explained. All testing is performed using the LGN data set described in the previous section.

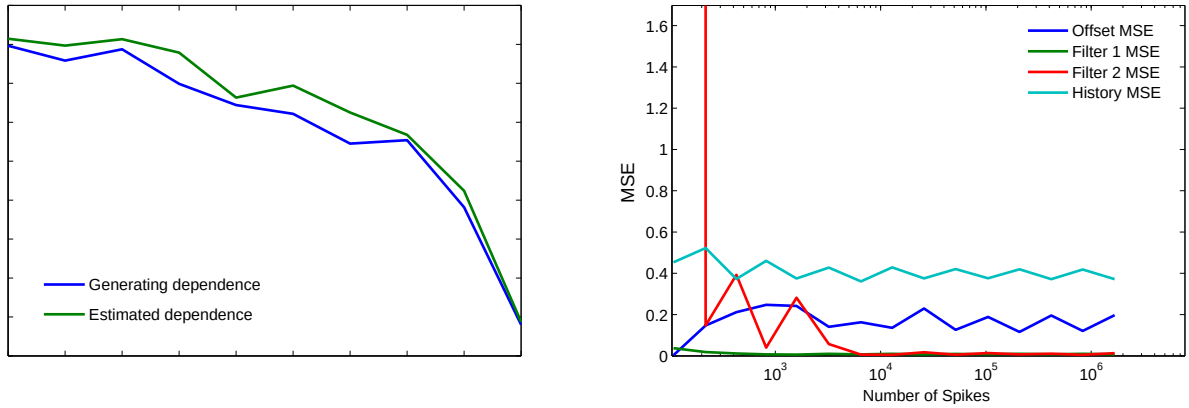

Figure 17: Left: The history dependence used to generate the spike train (along with one inhibitory and one excitatory filter), and the estimated dependence using the NIM for 10000 time bins. There were approximately 2000 spikes. Right: The mean square error between the generating parameters and the estimated parameters for increasing sample size, measured by the number of spikes present.

6.1 Cross Validation

The relative log likelihood is a measure of the amount of information a single spike possesses about the stimulus in our model. The measure used here uses the log-likelihood of the model LL_{model}, minus the log-likelihood of the "null" model LL_{null}, divided by the total number of spikes. The null model predicts a constant firing rate independent of the presented stimulus. This measure has a minimum value of 0, when the model only predicts a constant firing rate, and a maximum value of the single spike information (SSI) when the model predicts the correct firing rate for every time bin. Please see appendix C for a more detailed explanation.

To compute this measure we can use 10-fold cross validation on the dataset FFstim and FFspks for each model. To do this, we need to divide the dataset into 10 pieces and fit the parameters on 9 of the pieces, then find the relative log likelihood of the model using those parameters and the 10th piece of the dataset. We then repeat 9 more times, using each tenth of the dataset once for the validation. The average of these ten values is then used as the cross-validated relative log likelihood of the model.

For the GLM, GQM and NIM, when the upsampling factor is greater than 1 we also need to take regularization into account. We find the regularization constant by performing a round of 10-fold cross validation on each model using the FFstimR and FFspksR dataset. (This procedure is described in detail in the section on GLM regularization.) The constant that gives the highest relative log likelihood in this test is then used for the cross validation on the FFstim and FFspks dataset.

Additionally, the GQM and the NIM have optimal numbers of excitatory and inhibitory filters that in principle need to be found using different data. Due to the long computation times (table 2 took more than 100 hours to create), I simply used the optimal filter numbers that were found using FFspks and FFstim previously, as described in the section on GQM model selection.

One further complication that needs to be clarified is how we can determine the log likelihoods

of the STA and STC models. The same equation was used for these two models,

$$\mathcal{L}(\{r_{bin}(t)\}|\{n_t\}) = \sum_{t=1}^{T} n_t \log(r_{bin}(t)) - \sum_{t=1}^{T} r_{bin}(t),$$

again where $r_{bin}(t)$ is the probability of firing in each time bin given by the model and y_t is the observed spike counts in each bin. However, because the STA and STC utilize discrete nonlinear functions determined using the histogram method, there are occasions where there is not enough data to fully characterize the equation, especially for the 2-dimensional STC function. If there is a bin that is empty because there are no data points, I take the average of the neighbors to determine a value for that bin. In the STA these are the 2 neighbors on the sides, and in the STC these are the 8 surrounding neighbors (given that the bin is not on the boundary and that all neighbors have values; if this is not the case, then only the average of the neighbors that exist is used).

Figure 18 shows the results from the cross validation. One feature to note is that the models

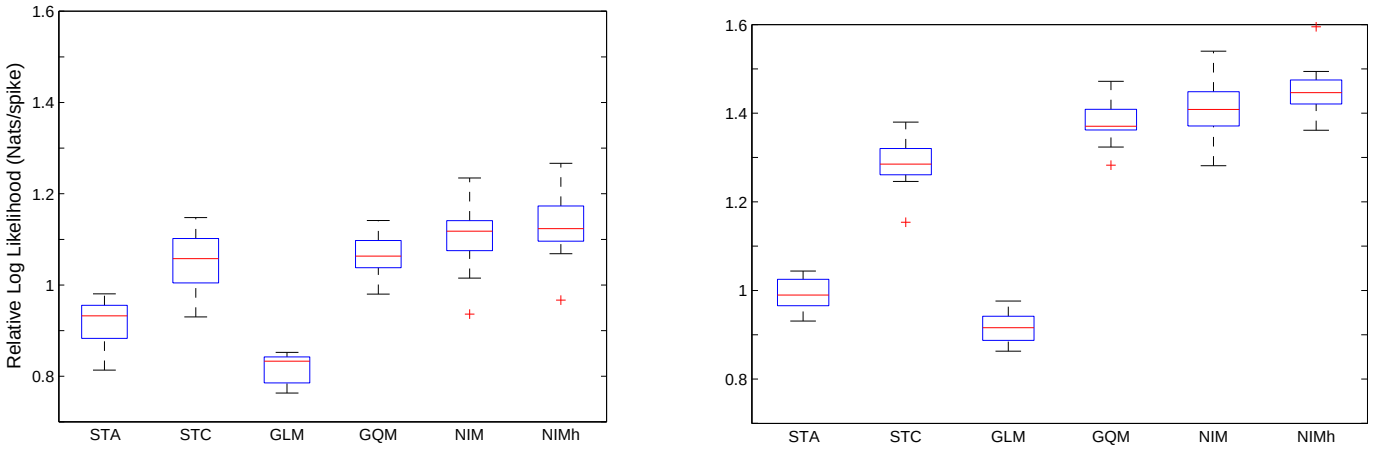

Figure 18: Left: Cross validation of the relative log likelihood (see appendix C) for the models considered in this paper, for an upsampling value of 1. NIMh refers to the NIM model with the history component added. Right: Comparison for an upsampling value of 2. For both plots, 10-fold cross validation was used, and 20 bins were used for the STA and STC discrete nonlinearities. For the right plot, all 4 of the MLE-based models used regularization, which was found searching through a hyperparameter space starting at 0 and going to 21 by increments of 3. The number of excitatory and inhibitory filters used for the GQM, NIM and NIMh models for both upsampling factors are the optimal combinations found by the BIC.

with the smallest RLLs are the linear models (STA and GLM); the remaining four nonlinear models perform substantially better. Also note that the added history component in the NIM has improved its performance, and we could expect similar results with the GLM and the GQM if the history term was included.

6.2 Fraction of Variance Explained

The fraction of variance explained (FVE) is a scalar metric that compares the mean square error of the model's predicted firing rate to the variance of the observed firing rate ([16]):

$$FVE = 1 - \frac{MSE(r_{model})}{Var(r_{obs})} \tag{26}$$

The simplest model of the neuron's firing rate predicts the average firing rate at each time bin;

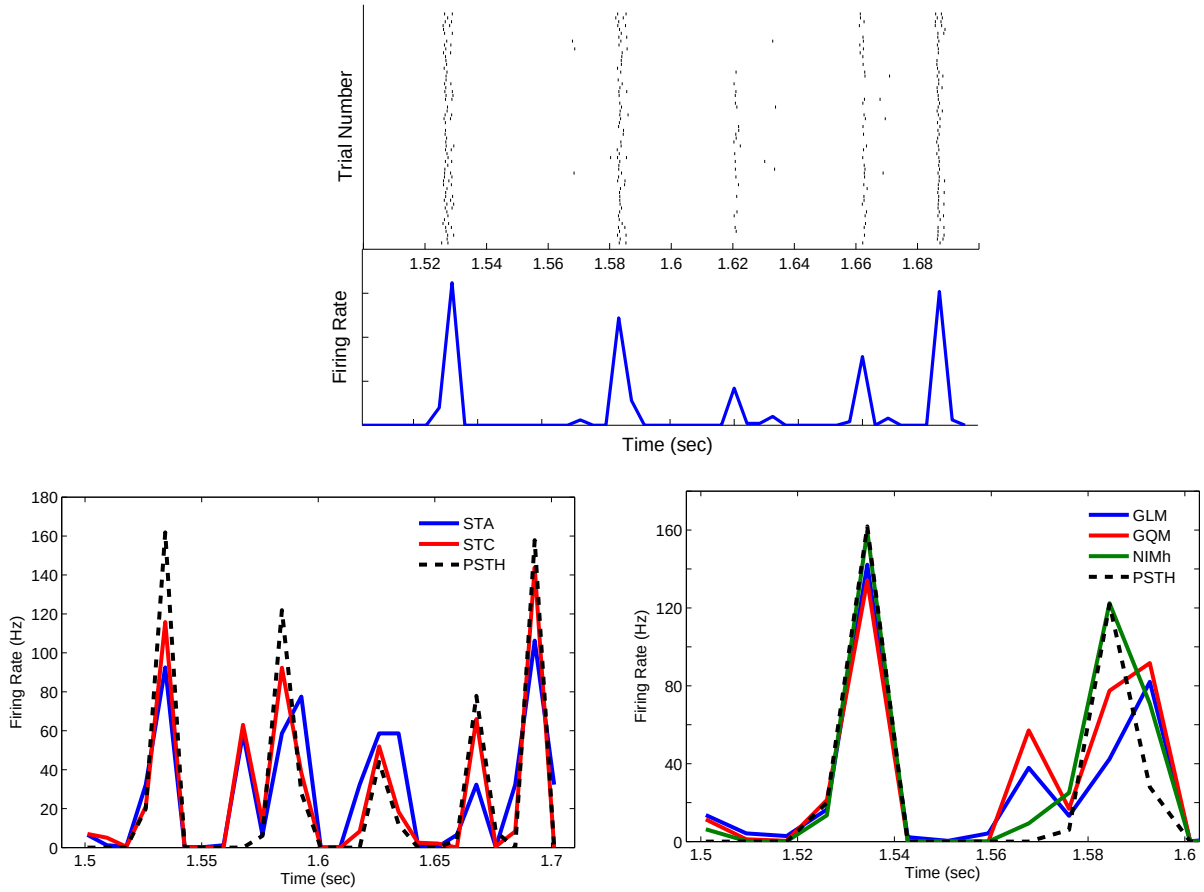

Figure 19: Top: Raster plot of a neuron's response to the same stimulus over 60 trials, along with the peri-stimulus time histogram underneath to approximate the neuron's firing rate. A bin size of 8.3 ms was used. Bottom Left: The rate predictions of the STA and STC, which were trained on a different set of data. Bottom Right: The rate predictions of the GLM, GQM and the NIM with the history component added. A zoomed in version is shown that displays an increase in activity that is particularly well described by all 3 models (the PSTH is the same as the rate predicted by the NIM) and a separate increase in activity that all 3 models have difficulty capturing.

the MSE of this model is then equal to the variance of the observed spike train, and the FVE is 0. On the other hand, if the model perfectly predicts the firing rate in each time bin, the mean square error of r_{model} is equal to zero and the fraction of variance explained is equal to 1.

Of course we cannot decipher the neuron's firing rate from a single spike train; instead we try to approximate it by presenting the same stimulus to the neuron over and over again, and recording the spiking pattern for all of these repeat trials, as shown at the top of figure 19(a). We can then divide the length of the experiment into time bins and approximate the probability of the neuron firing in a given time bin by looking at the proportion of repeat trials for which this neuron fired. This probability, multiplied by the size of the time bin, then gives the firing rate of the neuron during that time period. This is called the peri-stimulus time histogram, or PSTH for short.

The bottom half of figure 19(a) shows how this firing rate varies, and matches up with the aligned spikes above. The spikes were taken from 60 repeat trials of the same stimulus, and a time bin of width 8.3 ms was used. All of the models were fit using a separate set of stimulus/response data, and then these parameters were used to fit the firing rate of the neuron in response to this novel stimululs. Figure 19 shows how the moment-based estimators and the maximum likelihood-based estimators perform in matching this response. The FVE metric attempts to quantify this difference between the neuron's response and the models' ability to capture that response.

Figure 20 shows how the different models compare using this metric. The blue bars are for the case when the upsampling factor is 1, so that the parameters are fit using the same bin size that the PSTH is calculated with, namely 8.3 ms. For an upsampling factor of 2, this bin size is 4.15 ms. Notice that, unlike the relative log-likelihood metric, the FVE decreases for decreasing bin size. Intuitively this is because the models are better able to describe the neuron's behavior for coarse time intervals: at one extreme, if there is just one bin then the model only needs to be able to capture the average firing rate of the neuron to attain an FVE of 1, with no regard to temporal fluctuations of the rate that are driven by the stimulus.

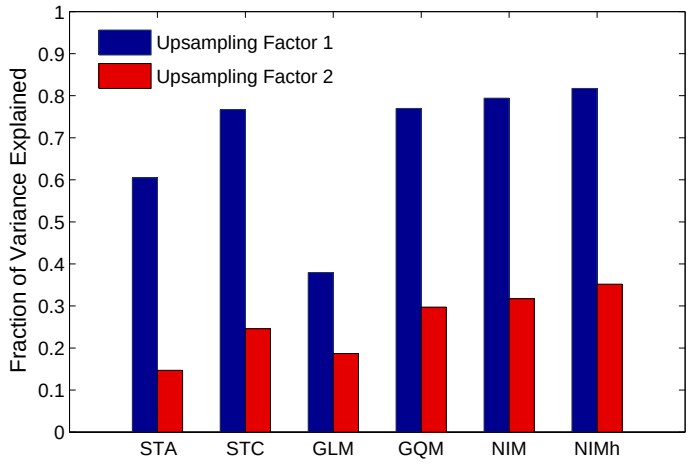

Figure 20: Fraction of variance explained by the various models for two time bin sizes.

7 Project Schedule and Milestones

The project schedule will be broken into two phases, roughly corresponding to the fall and spring semesters.

PHASE I - October through December

- (DONE) Implement and validate the STA (October)
- (MOVED) Develop code for gradient ascent method and validate (October)
- (DONE) Implement and validate the GLM with regularization (November-December)
- (DONE) Complete mid-year progress report and presentation (December)

PHASE II - January through May

- (DONE) Develop code for gradient ascent method and validate (January)
- (DONE) Implement and validate the STC (January-February)
- (DONE) Implement and validate the GQM (January-February)
- (DONE) Implement and validate the NIM with linear rectified upstream functions (March)
- (DONE) Develop software to test all models (April)
- (DONE) Complete final report and presentation

8 Deliverables

At the end of the year I will present the following deliverables:

- Implemented MATLAB code for all of the models STA, STC, GLM, GQM, NIM
- Implemented MATLAB code for the validation and testing of all the models
- Documentation of all code
- Results of validation and testing for all the models
- Final presentation and report

9 Conclusion

There have been many developments in neural modeling over the past two decades. Moving away from the biophysical models established by Hodgkin and Huxley has allowed researchers to ask new questions and discover new phenomena that might not otherwise have been possible.

The moment based estimators were at the forefront of this burgeoning interest in statistical descriptions of neurons. Their ease of implementation and use makes them easily accessible to researchers in the neuroscience field who might not have a strong background in mathematics. As

we have seen, the STC model is also quite competitive with its maximum likelihood counterparts. The main drawback to these models, however, are their need for spherically symmetric stimuli for fitting model parameters. Though corrections exist for other types of stimuli, these corrections begin to muddle the simple mathematics that make these models so attractive in the first place.

The introduction of the maximum likelihood based estimators puts the problem of model design in much richer framework of statistical rigor. Though we can see from the testing that the GLM behaves poorly even compared to the STA, the promise of the GLM is that its ideas can be extended to more powerful models like the GQM and the NIM. Along with the ability to specify particular classes of covariates in these models - including the stimulus, the spiking history, even the spiking history from other neurons - there are principled, well-developed methods to attach confidence bounds to our estimates, employ regularization, and explore other ideas in statistics that can lead to better models.

As experimental techniques in neuroscience progress, along with the ability to collect ever larger volumes of data, characterizing the responses of individual neurons is going to become an easier and arguably more interesting task in systems neuroscience. Hopefully the models can keep up.

A Validation of STA Histogram Method

The histogram method is a technique that can be used to find a discretized approximation to the nonlinear response function F that appears in the Linear-Nonlinear-Poisson (LNP) model. The algorithm works by dividing the range of the generator signal values $\mathbf{k} \cdot \mathbf{s}(t)$ in a number of bins. For each value of the generator signal we record which bin that value falls in and we also record whether or not there was a spike associated with that generator signal; the average number of spikes for generator signals that fall into a particular bin is then just the fraction of these two numbers:

$$\hat{F}(u_b) = \frac{\sum_t 1_{spike(t)} \cdot 1_{u_b < u(t) \leq u_{b+1}}}{\sum_t 1_{u_b < u(t) \leq u_{b+1}}}, \tag{27}$$

where $\hat{F}$ is the discrete approximation to the nonlinear response function, u is the generator signal $\mathbf{k} \cdot \mathbf{s}(t)$, and u_b is a generator signal whose value falls in bin b.

To validate my implementation of this algorithm, I first populate a generator signal vector with random Gaussian noise. For each value of the generator signal a corresponding spike is then inserted in the spike vector on a probabalistic basis. A random number *rand* is drawn from a uniform distribution, $rand \in U(0,1)$. Then we evaluate the Gaussian cumulative density function at the value of the generator signal u and determine the probability of a spike by

$$\mathbb{P}(spike) = \mathbb{P}(rand \in U(0,1) < cdf(u)|u) = cdf(u) = \int_{-\infty}^{u} \frac{1}{\sqrt{2\pi}} e^{-t^2/2} dt \tag{28}$$

The above algorithm for the histogram method, when run on this stimulus vector and associated spike vector, should produce the Gaussian cumulative density function. The proof of this statement is what follows, and is attributed to Dr. Radu Balan.

What we will show is that the expected value of the function $\hat{F}$ for the bin b is the Gaussian cdf with a small corrective term that is negligible under the circumstances. T will denote the length of the generator signal vector.

$$\begin{aligned} \mathbb{E}[\hat{F}(u_b)] &= \mathbb{E}\big[\mathbb{E}[\hat{F}(u_b)|k \text{ } signals \text{ } in \text{ } bin \text{ } b]\big] & (29) \\ &= \sum_{k=1}^{T} \mathbb{E}[\hat{F}(u_b)|k \text{ } signals \text{ } in \text{ } bin \text{ } b] \cdot \mathbb{P}(k \text{ } signals \text{ } in \text{ } bin \text{ } b) & (30) \end{aligned}$$

Let us look at the two parts of the term in the sum in equation 30. For the first part, the expected value of $\hat{F}$ given that k values of u fall into bin b can be found by looking at equation 27;

the bottom value is k, and the top number is the expected number of spikes:

$$\begin{aligned}
\mathbb{E}[\hat{F}(u_b)|k \textit{ signals in bin } b] &= \mathbb{E}\left[\frac{1}{k}\sum_{t_k=1}^{k} spike(t_k)\right] && (31)\\
&= \frac{1}{k}\sum_{t_k=1}^{k}\mathbb{E}[spike(t_k)] && (32)\\
&< \frac{1}{k}\sum_{t_k=1}^{k} cdf(u_{bM}) && (33)\\
&= \frac{1}{k}\sum_{t_k=1}^{k}\int_{-\infty}^{u_{bM}}\frac{1}{\sqrt{2\pi}}e^{-t^2/2}dt && (34)\\
&= \int_{-\infty}^{u_{bm}}\frac{1}{\sqrt{2\pi}}e^{-t^2/2}dt && (35)
\end{aligned}$$

where u_{bM} is the maximum value of the interval in bin b. For the second part, the probability that k values are in bin b, we can use the fact that the samples are independent of each other since they have been drawn independently from the Gaussian distribution. To find the probability that k values are in bin b we can then use the binomial distribution. If we use the notation

$$p = \mathbb{P}(u(t) \textit{ in bin } b) = \int_{u_{bm}}^{u_{bM}}\frac{1}{\sqrt{2\pi}}e^{-t^2/2}dt \tag{36}$$

again where u_{bM} is the maximum value of the interval in bin b and u_{bm} is the minimum value of the interval in bin b. Then

$$\mathbb{P}(k \textit{ signals in bin } b) = \binom{T}{k}p^k(1-p)^{T-k} \tag{37}$$

so that, when we put these two parts together, equation 30 becomes

$$\begin{aligned}
\mathbb{E}[\hat{F}(u_b)] &< \sum_{k=1}^{T} cdf(u_{bM})\cdot\binom{T}{k}p^k(1-p)^{T-k} && (38)\\
&= cdf(u_{bM})\sum_{k=1}^{T}\binom{T}{k}p^k(1-p)^{T-k} && (39)\\
&= cdf(u_{bM})\cdot(1-(1-p)^T) && (40)
\end{aligned}$$

In a similar manner we can establish a lower bound for $\mathbb{E}[\hat{F}(u_b)]$, and find that

$$\mathbb{E}[\hat{F}(u_b)] > cdf(u_{bm})\cdot(1-(1-p)^T). \tag{41}$$

Note that in practice the value of T, the total number of generator signals, is greater than 10,000, so that the term $(1-p)^T$ is essentially zero. Hence the expected value of the discrete approximation to F using the histogram method in this validation case should be the Gaussian cumulative density function.

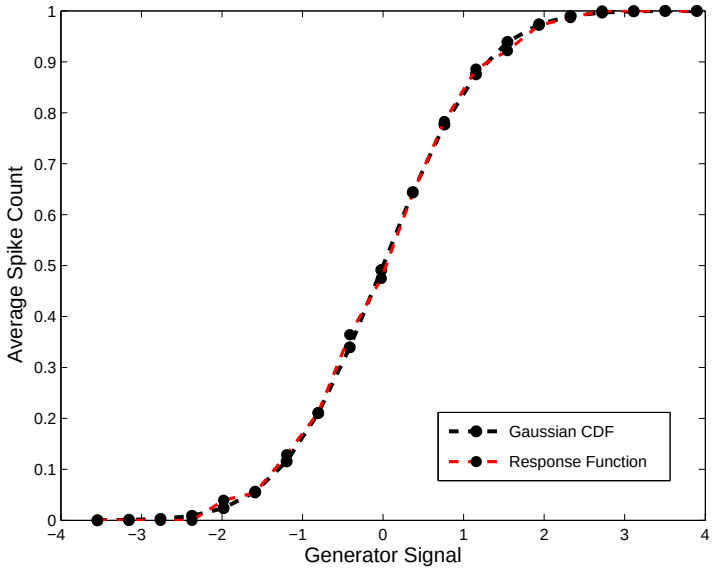

Figure 21: Validation results for the histogram method. The method should recover the Gaussian cdf in this test instance.

B Implementation of a quasi-Newton optimization algorithm

A quasi-Newton method can be implemented to solve unconstrained optimization problems, and is better suited for optimizing the log-likelihood function of the GLM than either the steepest descent method or the Newton method. The steepest descent method can easily get stuck in local minima and generally takes many iterations to converge. Newton's method, on the other hand, can find a minimum in many fewer iterations but at the cost of evaluating the Hessian, which takes an impractical amount of time for the log-likelihood function. Newton's method works by approximating the function at the current point with a second degree Taylor expansion, and choosing a descent direction that minimizes this quadratic approximation [18].

The vector form of a second degree Taylor expansion, centered at the point $\mathbf{x}_k$, is given by

$$f_T(\mathbf{x}_{k+1}) = f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)\Delta\mathbf{x} + \frac{1}{2}\Delta\mathbf{x}^T H \Delta\mathbf{x},$$

where $\Delta\mathbf{x} = \mathbf{x}_{k+1} - \mathbf{x}_k$. Taking the derivative of this expression with respect to $\Delta\mathbf{x}$ and setting the result equal to zero, the update equation for Newton's method is achieved:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - H^{-1}\nabla f(\mathbf{x}_k).$$

The descent direction, then, is $\mathbf{p}_k = -H^{-1}\nabla f(\mathbf{x}_k)$ and is generally calculated by solving the system of equations $H\mathbf{p}_k = -\nabla f(\mathbf{x}_k)$ rather than inverting the Hessian.

It is often the case that we are concerned not only with the descent *direction*, but with the magnitude of that step as well. The update equation then becomes

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$$

where α_k is a step size. Defining the proper size of α_k can be done in various ways, the most common of which is a set of conditions known as the *strong Wolfe conditions*, which state that

$$\begin{aligned} &1) \quad f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) \leq f(\mathbf{x}_k) + c_1 \alpha_k \mathbf{p}_k^T \nabla f(\mathbf{x}_k) \\ &2) \quad |\mathbf{p}_k^T \nabla f(\mathbf{x}_k + \alpha_k \mathbf{p}_k)| \leq c_2 |\mathbf{p}_k^T \nabla f(\mathbf{x}_k)|. \end{aligned}$$

The first condition ensures that the function value decreases sufficiently along the direction of $\mathbf{p}_k$, where "sufficiently" is governed by the value of c_1. A typical value given in reference [18] is 10^{-4}. The second condition ensures the slope is reduced sufficiently, and tends to keep the step size from becoming too small. Again this is governed by the value of c_2, a common value of which is 0.9 [18]. Various line search methods exist to find appropriate values of α_k, many of which can be found in reference [18].

The quasi-Newton method works the same as Newton's method, except instead of calculating the Hessian at each iteration it makes a low-rank update to an approximate Hessian. This approximation can be updated using several different formulas, and the particular implementation that I chose is called the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm. If $\mathbf{p}_k$ is descent direction on iteration k, α_k is the size of the step length, then

$$\begin{aligned} \mathbf{s}_k &= \alpha_k \mathbf{p}_k, \\ \mathbf{y}_k &= \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k), \end{aligned}$$

and the update formula is

$$B_{k+1} = B_k + \frac{\mathbf{y}_k \mathbf{y}_k^T}{\mathbf{y}_k^T \mathbf{s}_k} - \frac{B_k \mathbf{s}_k \mathbf{s}_k^T B_k}{\mathbf{s}_k^T B_k \mathbf{s}_k}, \tag{42}$$

where B_k is the approximate Hessian at iteration k.

Table 5 compares my own implementation of the BFGS quasi-Newton method with several of Matlab's built-in optimization routines - the steepest descent method, the BFGS quasi-Newton method, and the trust region method. My BFGS routine was able to find the correct minimum for three test functions, the 2D sphere function, the 2D Rosenbrock function, and the 2D Beale's function. Though the total number of iterations for my own routine is comparable to Matlab's quasi-Newton method, the number of iterations is far greater.

This higher number of iterations comes from my implementation of the line search method, which does not recycle evaluated function information as efficiently as Matlab's. Even still the times are comparable, mostly because these functions only have two parameters. As the number of parameters increases Matlab's routine begins to outperform my own.

The bottom of the table displays comparisons between Matlab's routines and my own when run on the log-likelihood function that must be minimized for the GLM. This information is for a filter of length 15, an upsampling factor of 1, and a regularization hyperparameter value of 1. The steepest descent routine performs surprisingly well, though this performance deteriorates rapidly for higher upsampling factors. Again my own BFGS implementation is comparable to Matlab's in terms of iterations, but still needs more function evaluations. The tolerance for all four routines was set to $1e^{-4}$, and figure 22 shows the filter differences between my implementation of the BFGS and the three Matlab routines.

2D sphere function - min: (0,0)			
Algorithm	Iterations	Func Evals	time (s)
Matlab Steepest Descent	1	2	0.0175
Matlab Quasi Newton	1	2	0.0100
Quasi Newton	2	7	0.0065
Matlab Trust Region	1	-	0.0372
2D Rosenbrock function - min: (1,1)			
Algorithm	Iterations	Func Evals	time (s)
Matlab Steepest Descent	2405	8444	4.4467
Matlab Quasi Newton	38	54	0.0766
Quasi Newton	40	164	0.0218
Matlab Trust Region	30	-	0.2055
2D Beale's function - min: (3,0.5)			
Algorithm	Iterations	Func Evals	time (s)
Matlab Steepest Descent	406	1017	0.5986
Matlab Quasi Newton	15	16	0.0543
Quasi Newton	14	58	0.0218
Matlab Trust Region	9	-	0.0763
Log-likelihood function			
Algorithm	Iterations	Func Evals	time (s)
Matlab Steepest Descent	14	88	1.5807
Matlab Quasi Newton	35	84	1.5620
Quasi Newton	39	134	3.0044
Matlab Trust Region	8	-	2.6880

Table 5: Comparison of my implementation of the quasi-Newton method using the BFGS update formula to several of Matlab's built-in routines for three test functions and the log-likelihood function. Note that Matlab output does not give function evaluations for the trust region method.

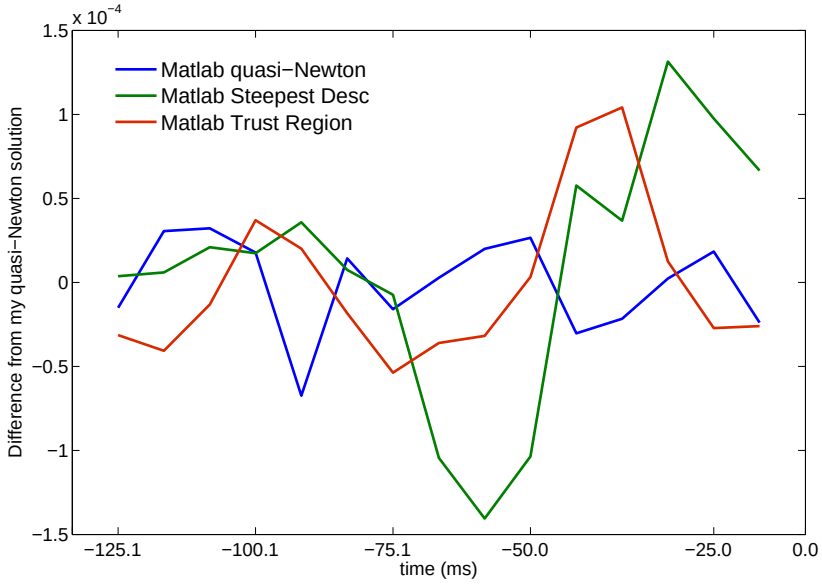

Figure 22: Values for the difference between the filter found by my own implementation of a quasi-Newton routine and the filter found by three of Matlab's built-in routines. The tolerance for all routines was set to $1e-4$; the filter was found through the GLM model with a filter length of 15 time steps, an upsampling factor of 1 and a regularization hyperparameter of 1.

C Relative Log Likelihood Measure

In comparing the various models in this report it will be instructive to look at a measure that is intimately related to the maximum likelihood, but compares each model with the simple model that predicts the average firing rate. This model will be referred to as the "null model", and is given by the total number of spikes divided by the number of time bins in the experiment. Thus this model treats each time bin as a Bernoulli trial with constant probability of seeing a spike, independent of time. Interestingly, this measure also happens to be linked to the Single Spike Information, a measure developed in reference [9], which is where the following derivation comes from.

The Single Spike Information (SSI) quantifies the amount of information that a single spike carries. To derive this quantity, we will consider discrete time bins of width Δt, and take Δt small enough that each time bin has either 0 spikes or 1 spike. The mutual information between a stimulus and a response used here was first proposed in reference [10]. We will use the notation $H[R]$ to denote the entropy of the response distribution, given by

$$H[R] = -\sum_r p(r)\log(p(r))$$

and $H[R|t]$ to represent the entropy of the response distribution at a particular point t in time.

$$H[R|t] = -\sum_r p(r|t)\log(p(r|t))$$

Reference [10] shows how this conditional entropy can be equivalent to the entropy of the response distribution given a particular stimulus. With this notation, the mutual information between the

stimulus (in this case time) and the response is

$$\begin{aligned} I(R,time) &= H[R] - \frac{\Delta t}{T}\sum_t H[R|t] \\ &= -\sum_r p(r)\log(p(r)) + \frac{\Delta t}{T}\sum_t\sum_r p(r|t)\log(p(r|t)) \end{aligned}$$

The first term on the right hand side represents the total information capacity of the response, and the second term represents the noise entropy, or the variability of the response for a given stimulus, averaged over all stimuli.

From here we want to find the mutual information between a single spike (the response) and the stimulus in a single time bin. In this case there are only two responses, 0 spikes or 1 spike. The relevant probabilities are:

$$\begin{aligned} p(r=1) &= \bar{r}\Delta t \\ p(r=0) &= 1-\bar{r}\Delta t \\ p(r=1|t) &= r(t)\Delta t \\ p(r=0|t) &= 1-r(t)\Delta t \end{aligned}$$

With these probabilities, and using the fact that $\bar{r} = \Delta t/T\sum_t^T r(t)$, the mutual information becomes

$$\begin{aligned} I(R,time) &= -\bar{r}\Delta t\log(\bar{r}\Delta t) - (1-\bar{r}\Delta t)\log(1-\bar{r}\Delta t) \\ &\quad +\frac{\Delta t}{T}\sum_t r(t)\Delta t\log(r(t)\Delta t) + \frac{\Delta t}{T}\sum_t(1-r(t)\Delta t)\log(1-r(t)\Delta t) \\ &= -\frac{\Delta t}{T}\sum_t r(t)\Delta t\log(\bar{r}\Delta t) - \frac{\Delta t}{T}\sum_t(1-r(t)\Delta t)\log(1-\bar{r}\Delta t) \\ &\quad +\frac{\Delta t}{T}\sum_t r(t)\Delta t\log(r(t)\Delta t) + \frac{\Delta t}{T}\sum_t(1-r(t)\Delta t)\log(1-r(t)\Delta t) \\ &= \frac{\Delta t}{T}\sum_t r(t)\Delta t\log\left(\frac{r(t)\Delta t}{\bar{r}\Delta t}\right) + \frac{\Delta t}{T}\sum_t(1-r(t))\Delta t\log\left(\frac{1-r(t)\Delta t}{1-\bar{r}\Delta t}\right) \end{aligned}$$

This is the mutual information between the stimulus and the response in a single bin of duration Δt. To find the single spike information, we divide this mutual information by the probability of seeing a spike, which is $\bar{r}\Delta t$ and get

$$I(R,time) = \frac{1}{T}\sum_t \frac{r(t)}{\bar{r}}\Delta t\log\left(\frac{r(t)\Delta t}{\bar{r}\Delta t}\right) + \frac{1}{T}\sum_t \frac{1-r(t)}{\bar{r}}\Delta t\log\left(\frac{1-r(t)\Delta t}{1-\bar{r}\Delta t}\right)$$

Now, in the limit as $\Delta t \to 0$, the second sum goes to zero and the first sum becomes an integral over time:

$$\begin{aligned} \lim_{\Delta t\to 0} I(R,time) &= \frac{1}{T}\int_0^T \frac{r(t)}{\bar{r}}\log\left(\frac{r(t)}{\bar{r}}\right)dt \\ &= \frac{1}{N_{spikes}}\int_0^T r(t)\log\left(\frac{r(t)}{\bar{r}}\right)dt \end{aligned}$$

The final equation is what is known as the single spike information, and has units of bits per spike if the logarithms are base 2. If the logarithms are natural logarithms with base e, the units become *nats* per spike. In discrete time, the SSI is

$$SSI = \frac{1}{N_{spikes}} \sum_t r(t)\Delta t \log\left(\frac{r(t)}{\bar{r}}\right). \tag{43}$$

Next we will see how this relates to the maximum likelihood. Let us assume that our model predicts the probability of spiking in a single time bin, i.e. $r(t)\Delta t = F(\mathbf{k} \cdot \mathbf{s}(t))$ in the case of the GLM. In the GLM section this quantity $r(t)\Delta t$ was referred to as r_{bin}, but to make notation simpler we will drop the Δt with the understanding that $r(t)$ is the probability of seeing a single spike in a bin at time t, and $1 - r(t)$ is the probability of seeing no spikes in a bin at time t. Furthermore, the observed rates $r_{obs}(t)$ will be restricted to the values 0 and 1, indicating that there was in fact a spike in that bin or there were no spikes.

The log-likelihood of this model, as given in equation (9), becomes

$$LL_{model} = \sum_t r_{obs}(t) \log(r(t)) - \sum_t r(t), \tag{44}$$

where LL_{model} refers to the fact that this is the log-likelihood of the full model. The log-likelihood of the null model is given by

$$\begin{aligned} LL_{null} &= \sum_t r_{obs}(t) \log(\bar{r}) - \sum_t \bar{r} \\ &= \log(\bar{r}) \sum_t r_{obs} - N_{spikes} \\ &= N_{spikes}(\log(\bar{r}) - 1) \end{aligned}$$

Now let us assume that our model is perfect, and that $r(t) = r_{obs}(t)$ for all time t. Then this equation becomes

$$\begin{aligned} LL_{obs} &= \sum_t r_{obs}(t) \log(r_{obs}(t)) - \sum_t r_{obs}(t) \\ &= \sum_t r_{obs}(t) \log(r_{obs}(t)) - N_{spikes} \end{aligned}$$

and, adding and subtracting a factor of $\sum_t r_{obs}(t) \log(\bar{r})$, this becomes

$$\begin{aligned} LL_{obs} &= \sum_t r_{obs}(t) \log\left(\frac{r_{obs}(t)}{\bar{r}}\right) + \sum_t r_{obs}(t) \log(\bar{r}) - N_{spikes} \\ &= N_{spikes} * SSI + N_{spikes}(\log(\bar{r}) - 1) \\ &= N_{spikes} * SSI + LL_{null} \\ \Rightarrow SSI &= \frac{LL_{obs}}{N_{spikes}} - \frac{LL_{null}}{N_{spikes}}. \end{aligned}$$

Now, since our model can never have a higher likelihood than the observations on which it is based, $LL_{model} \leq LL_{obs}$ and we arrive at

$$SSI \geq \frac{LL_{model}}{N_{spikes}} - \frac{LL_{null}}{N_{spikes}}. \tag{45}$$

We will now define the relative log likelihood (RLL) as

$$RLL = \frac{LL_{model}}{N_{spikes}} - \frac{LL_{null}}{N_{spikes}} \tag{46}$$

which is the log-likelihood per spike of our model minus the log-likelihood per spike of the null model. From equation 45, the RLL has an upper bound of the Single Spike Information, which is derived from the data itself. Therefore, the RLL of the "null model" is 0 and the RLL of the perfect model has a value of SSI. This allows us to compare different models, since the model with the highest value of $LL_{model} - LL_{null}$ per spike will best approximate the SSI. Furthermore, the units of this log-likelihood difference is given in bits/spike or nats/spike, depending on the logarithm used.

References

[1] Abbott, L.F. (1999). Lapicque's introduction of the integrate-and-fire model neuron (1907). *Brain Research Bulletin*, 50(5), 303-304.

[2] Hodgkin, A.L., Huxley, A.F. (1952). A quantitative description of membrane current and its application to conduction and excitation in the nerve. *The Journal of physiology, 117*, (4), 500-544.

[3] Chichilnisky, E.J. (2001). A simple white noise analysis of neuronal light responses. *Network: Comput. Neural Syst.*, 12, 199-213.

[4] Schwartz, O. et al. (2006). Spike-triggered neural characterization. *Journal of Vision*, 6, 484-507.

[5] Schwartz, O., Chichilnisky, E. J., and Simoncelli, E. P. (2002). Characterizing neural gain control using spike-triggered covariance. *Advances in neural information processing systems*, 1, 269-276.

[6] Paninski, L., Pillow, J., and Lewi, J. (2006). Statistical models for neural encoding, decoding, and optimal stimulus design.

[7] Shlens, J. (2008). Notes on Generalized Linear Models of Neurons.

[8] Paninski, L. (2004). Maximum Likelihood estimation of cascade point-process neural encoding models. *Network: Comput. Neural Syst.*, 15, 243-262.

[9] Brenner, N., Strong, S. P., Koberle, R., Bialek, W., and Van Steveninck, R. R. D. R. (2000). Synergy in a neural code. *Neural Computation*, 12(7), 1531-1552.

[10] Strong, S. P., Koberle, R., van Steveninck, R. R. D. R., and Bialek, W. (1998). Entropy and information in neural spike trains. *Physical review letters*, 80(1), 197.

[11] Mood, A., and Graybill, F. (1963). *Introduction to the Theory of Statistics*, New York, San Francisco, Toronto, London. McGraw-Hill.

[12] Park, I., Pillow, J. (2011). Bayesian Spike-Triggered Covariance Analysis. *Adv. Neural Information Processing Systems*, 24, 1692-1700.

[13] Rajan, K., Marre, O. and Tkacik, G. (2013). Learning quadratic receptive fields from neural responses to natural stimuli. *Neural computation*, 25.7, 1661-1692.

[14] Burnham, K. P., Anderson, D. R. (2004). Multimodel Inference: Understanding AIC and BIC in Model Selection. *Sociological Methods & Research* 33(2), 261-304.

[15] Butts, D. A., Weng, C., Jin, J., Alonso, J. M., and Paninski, L. (2011). Temporal precision in the visual pathway through the interplay of excitation and stimulus-driven suppression. *The Journal of Neuroscience* 31(31), 11313-11327.

[16] McFarland, J.M., Cui, Y., and Butts, D.A. (2013). Inferring nonlinear neuronal computation based on physiologically plausible inputs. *PLoS Computational Biology*.

[17] Dayan, P., Abbott, L. F. (2001). *Theoretical neuroscience* (Vol. 31). Cambridge, MA: MIT press.

[18] Nocedal, J., Wright, S.J. (2006). *Numerical Optimization*, Berlin, New York. Springer-Verlag.