Characterization of Nonlinear Neuron Responses
Mid Year 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

December 19, 2013

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, given a particular stimulus. This project will investigate some of the well known approaches, starting with simple linear models and progressing to more complex nonlinear models. One technique for fitting the parameters of the model uses the idea of a "linear receptive field" to characterize a feature subspace of the stimulus in which the neuron's response is dependent. Another parameter fitting technique that has been used with success is a likelihood-based method that can lead to efficient model parameter estimation. This project will examine both linear and nonlinear models, and use both the feature subspace technique and the maximum likelihood technique to fit the parameters of the appropriate model.

1 Introduction

A fundamental area of interest in the study of neurons is the relationship between the stimuli and the neural response. This relationship has been modeled using various approaches, from the estimation of the neuron's firing frequency given by the integrate-and-fire model [1] to the set of nonlinear differential equations given by the Hodgkin-Huxley model [2].

More recent approaches have utilized a probabalistic modeling framework for sensory neurons (neurons in the visual or auditory pathways, for example). Since techniques in neuroscience are now able to supply us with increasingly detailed physiological data, and since the nervous system itself is probabalistic, this statistical description of a neuron seems like a natural avenue of exploration [4]. The following sections detail both linear and nonlinear models, as well as two of the techniques used to estimate the parameters of the given models.

2 Linear Models

The goal of the linear models 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 us the probability of the neuron spiking [4].

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 Δ, then changes value instantaneously. This stimulus is represented by a stimulus vector $\bar{s}(t)$, where each entry is the pixel's value for a time duration of Δ. 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

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

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

2.1 The LNP using Spike-Triggered Average

The first way in which I 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 [7]:

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

where N is the number of spikes elicited by the stimulus and $\bar{s}(t_n)$ 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 average to zero), we can use the STA as the estimate for the linear filter $\bar{k}$ in the LNP model [7].

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 $\bar{k} \cdot \bar{s}(t)$ is divided into bins and the average spike count of each bin is calculated using $\bar{k}$, $\bar{s}(t)$ 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 $\bar{k}$ and using the resulting bin value to estimate the firing rate.

2.1.1 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 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.

The filter that is recovered by the STA has a resolution that is restricted

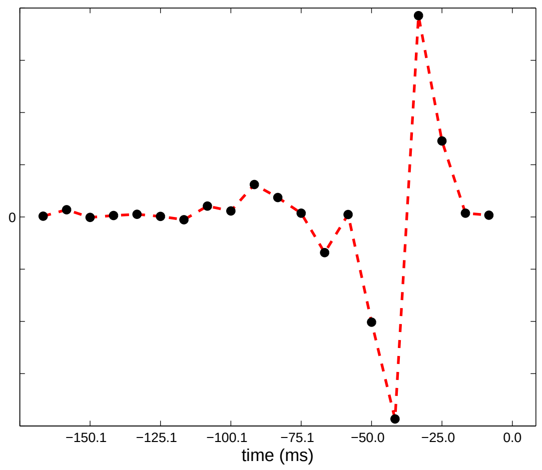

Figure 1: The recovered filter using the STA algorithm for a filter size of 20 time steps.

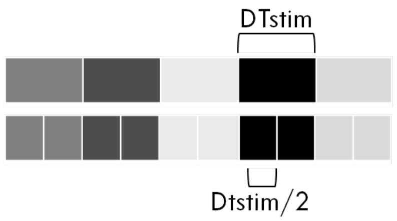

Figure 2: Upsampling the stimulus vector by a factor of 2.

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 $|\bar{s}|$ to a size of $n * |\bar{s}|$, 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 3.

As mentioned previously, the common approach to modeling the nonlin-

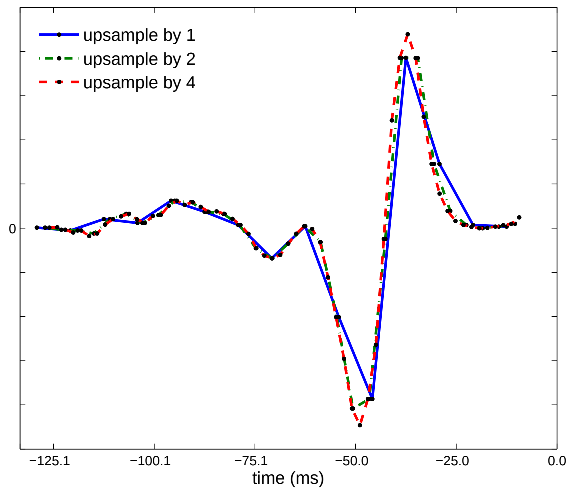

Figure 3: Upsampling the stimulus vector produces a filter with higher resolution. Filters shown are 15 original time steps long.

ear response function F in the literature is to use the "histogram method", which bins the values of the generator signal $(\bar{k} \cdot \bar{s}(t))$ and finds the average spike count for each bin. The result is a discretized version of the response function; given a certain signal $\bar{s'}(t)$, we can compute $\bar{k} \cdot \bar{s'}(t)$, find which bin this value 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 4.

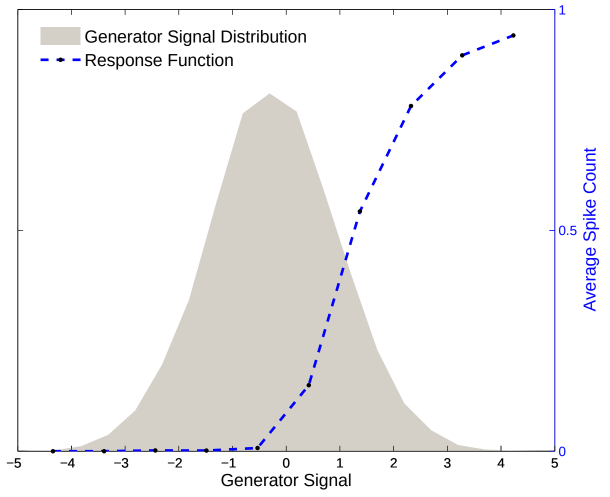

Figure 4: Plotting the nonlinear response function on top of the generator signal distribution shows the relation between the two, which is more important than the particular values each takes. This response function is for a filter of length 15 time steps and an upsampling factor of 1.

2.1.2 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. I first populate a stimulus vector with 15000 time samples drawn from a Gaussian distribution. I 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 I 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

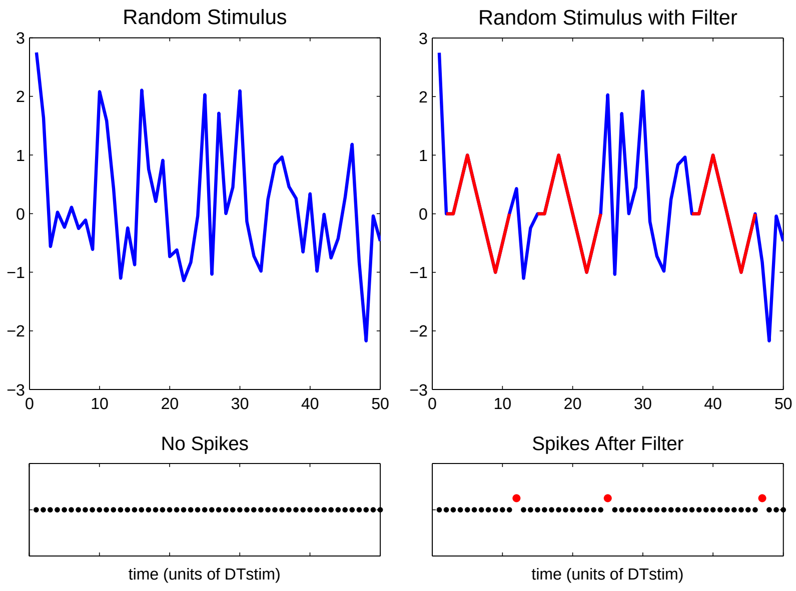

Figure 5: Visual representation of the creation of an artificial stimulus vector and corresponding spike vector for validation of the STA code.

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 I am 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

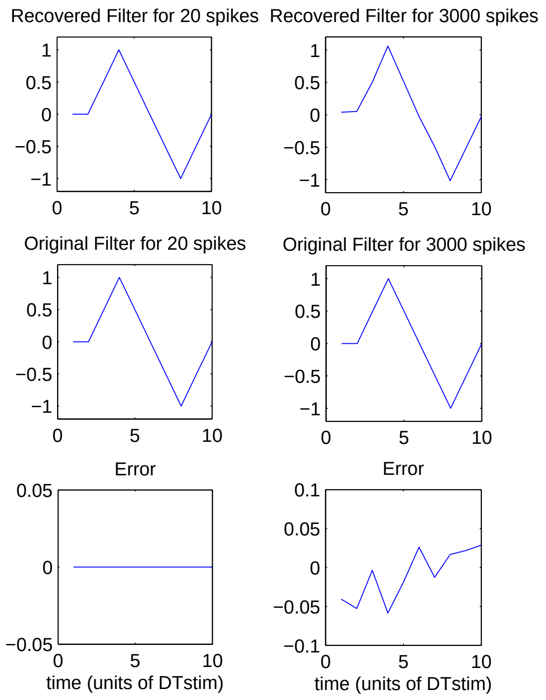

Figure 6: Top: Plot of the filter recovered by the STA program. Middle: Plot of the artificial filter created to validate the STA program. Bottom: Error between artificial filter and recovered filter. Artificial filter is 10 time steps long, with no upsampling.

the appendix.

2.2 The LNP using Maximum Likelihood Estimates (GLM)

The drawback to the STA technique is that it requires the stimulus data to be spherically symmetric. In order to make the LNP model more powerful we need to develop a new way to estimate the parameters of the model without this restriction. The Generalized Linear Model (GLM) is one such way to accomplish this task by using Maximum Likelihood Estimates to estimate the parameters of equation (1).

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

$$P(y|r(t)) = \frac{(\Delta r(t))^y}{y!} e^{-\Delta r(t)}. \tag{3}$$

If we are now interested in observing an entire spike train Y, which is a vector of (assumed independent) spike counts y_t binned at a time resolution of Δ, the conditional probability of this event is given by

$$P(Y|\{r(t)\}) = \prod_{t=1}^{T} \frac{(\Delta r(t))^{y_t}}{y_t!} e^{-\Delta r(t)} \tag{4}$$

where the product runs over each element of the spike count vector Y, from time 1 to time T, and $\{r(t)\}$ is the collection of rates, one for each element in Y. Normally we would view this equation as the probability of the event Y 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 Y (and that we are using a Poisson distribution)? Viewed in this way equation (4) becomes a function of the collection of rate parameters $\{r(t)\}$, and is known as the *likelihood* function [5]:

$$P(\{r(t)\}|Y) = \prod_{t=1}^{T} \frac{(\Delta r(t))^{y_t}}{y_t!} e^{-\Delta r(t)}. \tag{5}$$

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 Y, 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 (5) becomes

$$\mathcal{L}(\{r(t)\}|Y) = \sum_{t=1}^{T} y_t log(r(t)) - \Delta \sum_{t=1}^{T} r(t). \tag{6}$$

At this point we have an optimization problem to solve involving the linear filter $\bar{k}$ and the nonlinear function F. Fortunately, it has been shown by Paninski in [6] 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 ([6],[9],[11]) 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 $\bar{k}$ 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.

2.2.1 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.

Once the log-likelihood function has been coded the GLM implementation simply reduces to an unconstrained optimization problem. In my initial project schedule I intended to write my own optimization routine. I started with a gradient descent method, which proved to work but took too much time to be practical. I then decided to code 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. At this point I decided that my time would be better spent moving forward with the GLM, and since then I have been using Matlab's *fminunc* routine. During winter break I will return to this optimization routine and produce a quasi-Newton method that should be more efficient than either of my previous attempts.

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 $F(u) = log(1 + exp(u - c))$, where c is a parameter of the function. Using this form the complete log-likelihood function (and objective function of the optimization problem) becomes

$$\mathcal{L}(\{r(t)\}|Y) = \sum_{t=1}^{T} y_t log(log(1 + e^{\bar{k}\cdot\bar{s}(t)-c})) - \Delta \sum_{t=1}^{T} log(1 + e^{\bar{k}\cdot\bar{s}(t)-c}). \quad (7)$$

The optimization routine will find the (global) maximum log-likelihood, which will be at particular values for $\bar{k}$ and c. 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

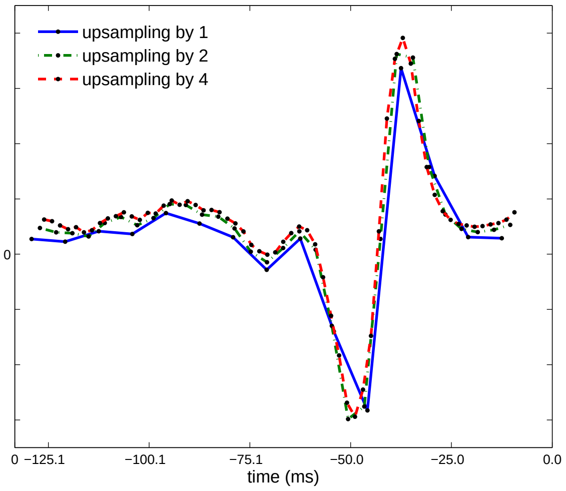

Figure 7: Filters found using the GLM for various upsampling factors.

parameters; in this case the function offset c. Like the STA, it is more instructive to view the resulting function relative to the distribution of the generator signal, as shown in figure 8.

2.2.2 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 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 log-likelihood function then

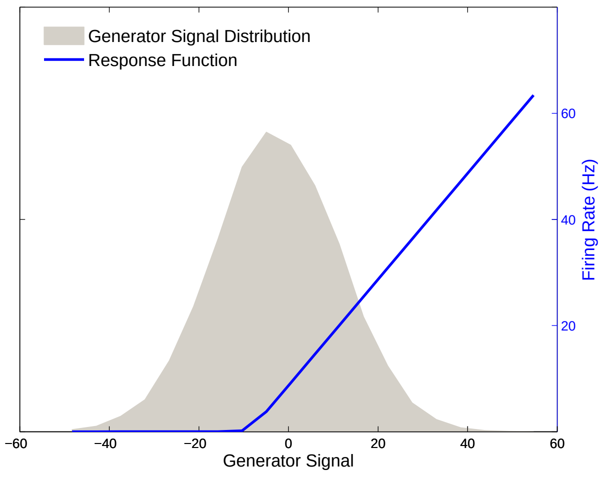

Figure 8: Response function for a filter of length 15 time steps and an upsampling factor of 1.

becomes

$$\mathcal{L}(\{r(t)\}|Y) = \sum_{t=1}^{T} y_t log(r(t)) - \Delta \sum_{t=1}^{T} r(t) - \lambda \|L^t \bar{k}\|_2^2, \tag{8}$$

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 we can employ a technique known as *nested cross validation*; we choose a value for λ and fit the model on some percentage of the test data (80% in the plots that follow). We then use the remaining percentage of the test data (20%) to determine how well the model generalizes to novel data by computing the associated log-likelihood value. The model parameters are only optimized once for each value of the regularization hyperparameter; admittedly, in a more complete analysis, a full 5-fold cross validation will need to be performed, averaging the log-likelihood value over the 5 different validation sets. I will include this more complete analysis in my final report.

This process is now repeated for different values of λ and the value that produces the largest value of the log-likelihood function 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.

Figure 9(a) shows the initial increase in both the value of the log-likelihood function for the data that the model was fitted on (blue line) and the novel data that the model was tested on (green line), for an upsampling factor of 2. The λ value for which the model maximizes the log-likelihood is at $\lambda = 700$.

Figure 9(b) shows a similar plot for when the stimulus has been up-

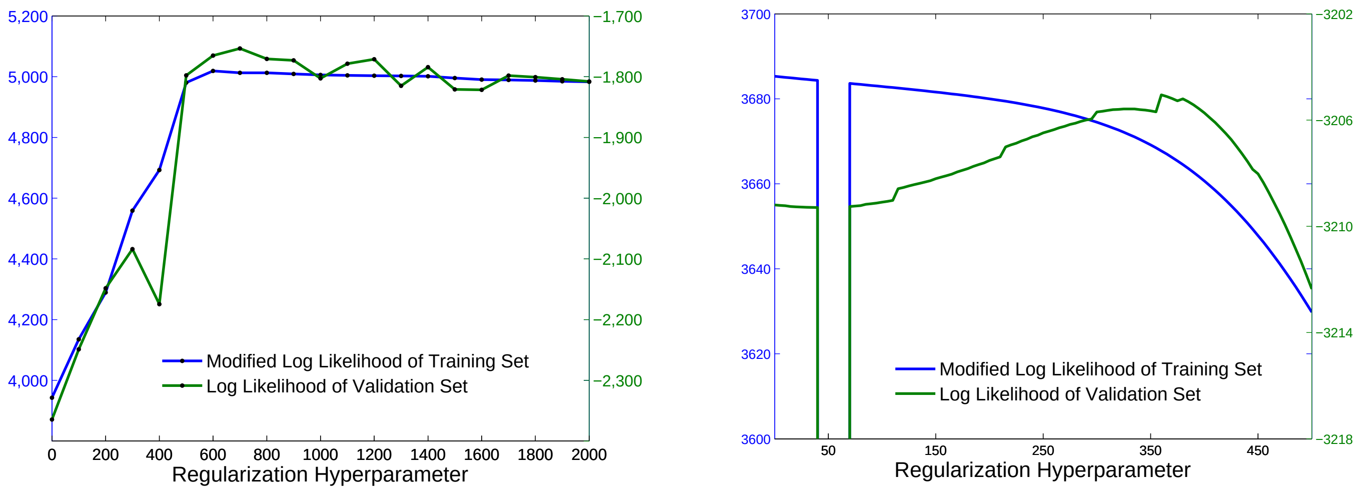

Figure 9: Results from the nested cross validation of the regularization hyperparameter. Right: Upsampling factor of 2. Left: Upsampling factor of 4. Both plots are for a filter length of 15 time steps.

sampled by a factor of 4. Here it is easier to see evidence of overfitting. The likelihood for the data that the model was fitted on starts at a maximum value, indicating a good fit, and decreases as we increase the regularization parameter, indicating an increasingly worse fit. However, the likelihood for the model on novel data is increasing during this same interval, indicating that the model was initially overfitting the data is now able to generalize better. At $\lambda = 360$ the log-likelihood value reaches a maximum.

Figures 10(a) shows the effect of regularization on the filter for an upsampling factor of 2. For $\lambda = 0$ the filter is not smooth, but becomes much more so at $\lambda = 700$. When $\lambda = 10000$ the effects of too much smoothing are evident. Figure 10(b) shows the filter for different λ values when the stimulus is upsampled by a factor of 4.

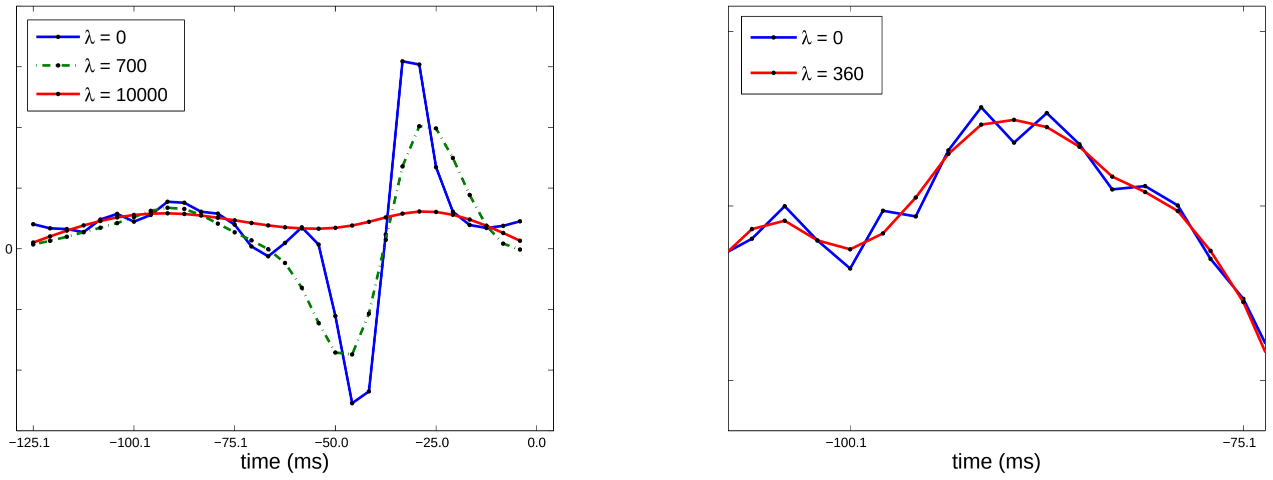

Figure 10: Resulting filters after regularization term has been included. Right: Filters for 3 different values of the hyperparameter, showing its increasing effect on the smoothness of the filter. Stimulus has been upsampled by a factor of 2. Left: Stimulus has been upsampled by a factor of 4. Filters for $\lambda = 0$ and $\lambda = 360$ look similar at the normal scale, but the effect of the regularization term is clearly seen upon closer inspection.

2.2.3 GLM Validation

The validation of the GLM rests on making sure that the optimization routine is working properly. At this point, since I am using a built-in Matlab function, there is nothing to validate for the GLM. This section will be updated in the final report with the performance of my own optimization routine.

This is as far as I have gotten in my project for the fall semester. During the spring semester I will resume my project starting with the STC and then move on to the GQM and NIM.

2.3 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.

The STA can be interpreted as the difference between the means of the raw stimulus data and the spike-triggering stimulus data. 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:

$$STC = \frac{1}{N-1} \sum_{n=1}^{N} \left[\bar{s}(t_n) - STA\right] \left[\bar{s}(t_n) - STA\right]^T \tag{9}$$

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. For the purpose of this project I 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

$$r(t) = F(\bar{k}_e \cdot \bar{s}(t), \bar{k}_i \cdot \bar{s}(t)) \tag{10}$$

where $\bar{k}_e$ and $\bar{k}_i$ denote the excitatory and inhibitory filters, respectively. 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 $(\bar{k}_e \cdot \bar{s}(t), \bar{k}_i \cdot \bar{s}(t))$ 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 Nonlinear Models

What makes the linear models 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.

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

$$r(t) = F\big(f_1(\bar{k_1} \cdot \bar{s}(t)), f_2(\bar{k_2} \cdot \bar{s}(t)), \ldots, f_n(\bar{k_n} \cdot \bar{s}(t))\big) \tag{11}$$

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 [11]; 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

$$r(t) = F\Bigg(\sum_i f_i(\bar{k_i} \cdot \bar{s}(t))\Bigg). \tag{12}$$

The next two sections will examine particular instances of equation (12) and how the parameters are fitted.

3.1 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

$$r(t) = F\big(\frac{1}{2}\bar{s}(t)C\bar{s}(t) + \bar{b}^T\bar{s}(t) + a\big), \tag{13}$$

where C is a symmetric matrix, $\bar{b}$ is a vector, and a is a scalar [9]. 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, $\bar{b}$ and a.

3.2 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 [11]. The equation, similar to equation (12), is given by

$$r(t) = F\Bigg(\sum_i w_i f_i(\bar{k_i} \cdot \bar{s}(t))\Bigg), \tag{14}$$

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 $\bar{s}(t)$ is projected onto various subspaces by the filters k_i; the functions f_i transform these projections nonlinearly, and the results are summed 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 I 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 [11]. With these assumptions made, the gradient ascent routine will only need to optimize the filters k_i.

4 Databases & Implementation

The initial dataset that I will use 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. Along with this is a vector that holds the time in seconds at which a spike was recorded. The second part of the data consists of a 10 second stimulus, again changing every 0.0083 seconds, and 64 separate trials during which spike times were recorded.

Additional datasets for testing and validation will be simulated data from other single neuron models. In general the models that I will be developing will require fitting one or two filters, each of which can contain up to 100 parameters that need to be fitted.

All software development will be performed on my personal computer, a Dell Inspiron 1525 with an Intel Core 2 Duo processor and 3GB of RAM. The programming language will be MATLAB.

5 Testing

Testing of the various models will be performed using two metrics: k-fold cross validation and fraction of variance explained.

k-fold cross-validation will be used on the log-likelihood of the model LL_x, minus the log-likelihood of the "null" model LL_0. The null model predicts a constant firing rate independent of the presented stimulus. This testing will be performed on all models at the end of the project.

The fraction of variance explained is a scalar metric that compares the mean square error of the model R to the variance of the output Y:

$$FVE = 1 - \frac{MSE(R)}{Var[Y]} \tag{15}$$

The simplest model of R is the constant function equal to the mean of Y. In this case the mean square error of R is equal to the variance of Y, and the fraction of variance explained by our naive model is zero. However, if our model perfectly predicts the values of Y, the mean square error of R is equal to zero and the fraction of variance explained is equal to one.

6 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 LNP model using the STA (October)
- (PENDING) 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

- (New Item) Develop code for gradient ascent method and validate (January)
- Implement and validate the LNP model using the STC (January-February)

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

7 Deliverables

At the end of the year I will be able to present the following deliverables:

- Implemented MATLAB code for all of the models - LNP-STA, LNP-GLM, LNP-STC, 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
- Mid-year presentation and report
- Final presentation and report

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 $\bar{k} \cdot \bar{s}(t)$ 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:

$$\hat{F}(u_b) = \frac{\sum_t 1_{spike(t)} \cdot 1_{u_b < u(t) \leq u_{b+1}}}{\sum_t 1_{u_b < u(t) \leq u_{b+1}}}, \tag{16}$$

where $\hat{F}$ is the discrete approximation to the nonlinear response function, u is the generator signal $\bar{k} \cdot \bar{s}(t)$, 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

$$\mathbb{P}(spike) = \mathbb{P}(rand \in U(0,1) < cdf(u)|u) = cdf(u) = \int_{-\infty}^{u} \frac{1}{\sqrt{2\pi}} e^{-t^2/2} dt \tag{17}$$

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.

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.

$$\begin{aligned} \mathbb{E}[\hat{F}(u_b)] &= \mathbb{E}\big[\mathbb{E}[\hat{F}(u_b)|k \ signals \ in \ bin \ b]\big] &(18)\\ &= \sum_{k=1}^{T} \mathbb{E}[\hat{F}(u_b)|k \ signals \ in \ bin \ b] \cdot \mathbb{P}(k \ signals \ in \ bin \ b) &(19)\end{aligned}$$

Let us look at the two parts of the term in the sum in equation 19. 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 16; the bottom value is k, and the top number is the expected number of spikes:

$$\begin{aligned}
\mathbb{E}[\hat{F}(u_b)|k\ signals\ in\ bin\ b] &= \mathbb{E}\left[\frac{1}{k}\sum_{t_k=1}^{k} spike(t_k)\right] && (20)\\
&= \frac{1}{k}\sum_{t_k=1}^{k} \mathbb{E}[spike(t_k)] && (21)\\
&< \frac{1}{k}\sum_{t_k=1}^{k} cdf(u_{bM}) && (22)\\
&= \frac{1}{k}\sum_{t_k=1}^{k} \int_{-\infty}^{u_{bM}} \frac{1}{\sqrt{2\pi}} e^{-t^2/2} dt && (23)\\
&= \int_{-\infty}^{u_{bm}} \frac{1}{\sqrt{2\pi}} e^{-t^2/2} dt && (24)
\end{aligned}$$

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

$$p = \mathbb{P}(u(t)\ in\ bin\ b) = \int_{u_{bm}}^{u_{bM}} \frac{1}{\sqrt{2\pi}} e^{-t^2/2} dt \qquad (25)$$

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

$$\mathbb{P}(k\ signals\ in\ bin\ b) = \binom{T}{k} p^k (1-p)^{T-k} \qquad (26)$$

so that, when we put these two parts together, equation 19 becomes

$$\begin{aligned}
\mathbb{E}[\hat{F}(u_b)] &< \sum_{k=1}^{T} cdf(u_{bM}) \cdot \binom{T}{k} p^k (1-p)^{T-k} && (27)\\
&= cdf(u_{bM}) \sum_{k=1}^{T} \binom{T}{k} p^k (1-p)^{T-k} && (28)\\
&= cdf(u_{bM}) \cdot (1-(1-p)^T) && (29)
\end{aligned}$$

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

$$\mathbb{E}[\hat{F}(u_b)] > cdf(u_{bm}) \cdot (1 - (1-p)^T). \tag{30}$$

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.

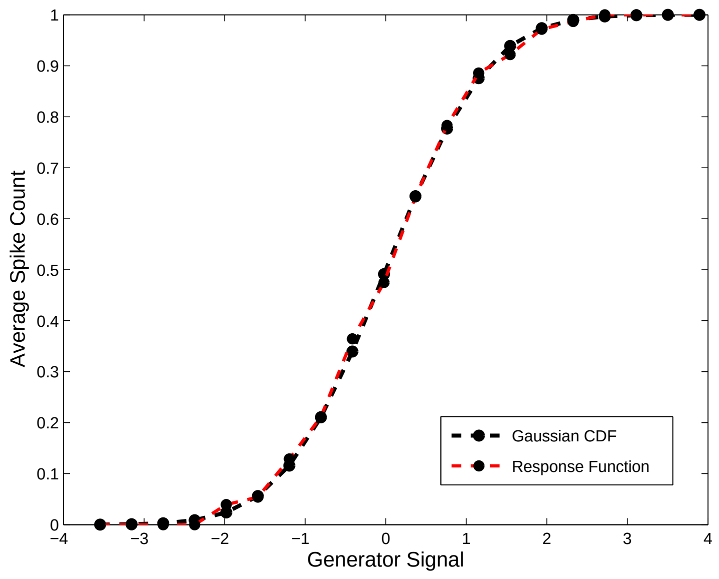

Figure 11: Validation results for the histogram method. The method should recover the Gaussian cdf in this test instance.

References

[1] Abbott, L.F. (1999). Lapicque's introduction fo the integrate-and-fire model neuron (1907). *Brain Research Bulletin, Vol. 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] Paninski, L., Pillow, J., and Lewi, J. (2006). Statistical models for neural encoding, decoding, and optimal stimulus design.

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

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

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

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

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

[10] Butts, D. A., Weng, C., Jin, J., Alonso, J. M., & 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.

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