Linear Nonlinear Poisson Model

Jong Woo Nam

March 25, 2020

1 Introduction

This document explains how the **make_design_matrix.m** and **neg_log_likli_poisson.m** are coded in detail with some background on how the problem is mathematically formulated.

2 Mathematical Formulation

2.1 Log Likelihood Function

The Linear-Nonlinear-Poisson model fits the spatio-temporal receptive fields to maximize the likelihood of the observed data.

Since likelihood function involves multiplication of all of the observed data, which is often computationally expensive, the log of the likelihood is more often used, converting multiplication to summation.

The log likelihood function for the LNP model is defined by the following equation.

$$L(\theta) = \sum_{t=1}^{T} (y_t log(\Delta f(\vec{s}_t \cdot \vec{w}(\theta)) - \Delta f(\vec{s}_t \cdot w(\vec{\theta})) - log(y_t!)) \tag{1}$$

where y_t is the number of spikes at time period t, Δ is the time delta between each stimulus (or the time delta used for binning the spikes), $\vec{s}_t$ is the stimulus vector (flattened) presented at time period t, $\vec{w}$ is the weights (same size as $\vec{s}_t$), and θ is the parameters subjected to optimization.

The confusing bit here is the difference between θ and $\vec{w}$. To understand the difference, let's take a look at how the stimulus vector $\vec{s}_t$ is constructed.

$\vec{s}_t$ is not only the stimulus (the frame) presented at time period t, but is a collection of the stimuli leading up to time t that could affect the response of the spiking of the neuron at t. Thus, it is a three-dimensional array, where the first two dimensions are the frame sizes, and the third dimension is the temporal dimension. Usually the size of the third dimension is user-defined, but often values consistent with the biophysical mechanics of the neurons are used. For instance, I used 25, which is in units of frames translating to 400ms given the frame rate of 60Hz.

This three dimensional array is then flattened to make the vector $\vec{s}_t$. Consequently, to make the dot product possible, $\vec{w}$ has to have the same size as the $\vec{s}_t$. Weight $\vec{w}$ is thus what gets multiplied to the stimulus, the three-dimensional spatio-temporal receptive field.

On the other hand, θ is the parameter vector getting the optimized. What θ is depends on the constraints you put for the optimization. For instance, in case of space-time nonseparable where you have to optimize each of w_i values separately, θ and $\vec{w}$ are the same.

However, if you assume that the spatial and temporal receptive fields are separable, $\vec{w}$ is a function of θ, where $w_i = \theta_t * \theta_s$ for the corresponding temporal rf value θ_t and the spatial rf value θ_s.

This is the reason why the log likelihood function is written as a function of θ not $\vec{w}$; you allow the optimizer to do gradient-ascent separately on each of the given θ_is, not the w_is.

2.2 Optimization / Computing gradient

In order for the matlab's optimization functions such as *fminunc* to work, you need to provide it with a loss function subjected to optimization. The loss function needs to give out 1) the current loss value, and 2) the gradient. (Hessian, or the second derivative, is optional but if provided, it can accelerate the optimization).

The gradient can be computed by taking partial derivatives in terms of each of the θ_is and concatenating them into a vector.

In simplest space-time separable case, the partial derivative looks like the following:

$$\frac{\partial L}{\partial \theta_i} = \sum_{t=1}^{T} [(y_t - \Delta exp(\vec{s}_t \cdot \vec{w})) * \vec{s}_t \cdot \frac{\partial \vec{w}}{\partial \theta_i}] \tag{2}$$

Here, the non-linearity function f is set to be an exponential function, denoted as *exp*. Note that this corresponds to the link function for poisson distribution in the Generalized Linear Model (GLM).

Since $\vec{w}$ and θ are the same for the nonseparable case, $\frac{\partial \vec{w}}{\partial \theta_i}$ is just a vector with one '1' in $i^t h$ position.

For other more constrained cases, computing $\frac{\partial \vec{w}}{\partial \theta_i}$ becomes more difficult. For these cases, I will explain how I did it along with the code in section 3.

2.3 Hints for extending the likelihood formulation with different constraints

We can generalize and make our model fit for more flexible conditions.

For instance, let's say we want to fit two spatial receptive fields, one for the on stimulus and another for the off stimulus. Just carefully follow the steps in creating the likelihood function and the gradient:

$$L(\theta) = \sum_{t=1}^{T} (y_t log(\Delta f(\vec{s}_t^{on} \cdot \vec{w}^{on} + \vec{s}_t^{off} \cdot \vec{w}^{off}) - \Delta f(\vec{s}_t^{on} \cdot \vec{w}^{on} + \vec{s}_t^{off} \cdot \vec{w}^{off}) - log(y_t!)) \tag{3}$$

where $\vec{s}_t^{on}$ is the on stimulus where anything below zero were replaced with zero, and $\vec{s}_t^{off}$ is the off stimulus. $\vec{w}^{on}$ and $\vec{w}^{off}$ are the corresponding on and off receptive field weights.

Computing the gradient:

$$\frac{\partial L}{\partial \theta_i} = \sum_{t=1}^{T} [(y_t - \Delta exp(\vec{s}_t \cdot \vec{w})) * (\vec{s}_t^{on} \cdot \frac{\partial \vec{w}^{on}}{\partial \theta_i} + \vec{s}_t^{off} \cdot \frac{\partial \vec{w}^{off}}{\partial \theta_i})] \tag{4}$$

Again, the partial derivatives we need to compute are not trivial, but I will explain how those are done in later section.

Though not done here, one can see how if the nonlinearity f is also parameterized and optimized with the receptive field, it can be done pretty easily. The gradient would be little harder to derive, since f will be a function of θ too.

3 How the code is written

With understanding of the math behind the code, let's look at how the code is written.

3.1 make_design_matrix.m

Design matrix is basically the three-dimensional stimulus arrays for each time period t, flattened and concatenated in a row.

sn_stim is a three-dimensional array with all of the frames concatenated along the third dimension. Thus, to create a row in the design matrix, we need to first get the three-dimensional stimulus array. This can be done by cutting the **sn_stim** along the third dimension with the wanted receptive field's temporal length (**rf_temporal_len**).

```
 1     function xdsgn = make_design_matrix(sn_stim, rf_temporal_len)
 2 %make_design_matrix: creates design matrix which has dimensions of nxm (rxc),
 3 %where n = total number of stimulus (time length of the recording),
 4 %and m = size of the receptive field (gridsize^2 * rf temporal length)
 5     stim_len = size(sn_stim.stimulus,3);
 6     % pad with zeros at the start
 7     paddedStim = cat(3, zeros(sn_stim.config.gridsize,sn_stim.config.gridsize,
           rf_temporal_len-1), sn_stim.stimulus);
 8     xdsgn = zeros(stim_len,sn_stim.config.gridsize^2*rf_temporal_len);
 9     % loop through the stimulus with jump size of single frame and flatten
10     % to a row in the design matrix
11     for j = 1:stim_len
12         xdsgn(j,:) = reshape(paddedStim(:,:,j:j+rf_temporal_len-1),[1,rf_temporal_len*
               sn_stim.config.gridsize^2]);
13     end
14 end
```

line 7:

To match the size of the resulting design matrix with the original temporal length of the stimulus, we need to front-pad the stimulus matrix first. Line 7 is padding the stimulus matrix with a three dimensional zero-matrix where the temporal (third dimension) length is **rf_temporal_len** $- 1$.

line 11 - 13:

Here, we are cutting slices out of the stimulus matrix, with step size of 1 along the third, temporal axis. Then in line 12, the sliced matrix gets flattened to a row.

3.2 neg_log_likli_poisson.m

This is the function that computes the log likelihood (negative log-likelihood, since we are using fminunc which minimizes the loss function), as well as the gradient.

The main function can currently deal with three different cases: time-space non-separable case, time-space separable case, and time-space and on-off separable case.

```
function [neg_log_likelihood, dL] = neg_log_likli_poisson(stim, spike_train, glm_weights, td, temporal_len, grid_size)
%LOG_LIKLI_POISSON computes log likelihood of the spike train given the
%weights being optimized with glm. Uses exponential nonlinearity.
%Handles three different cases: 1) time-space nonseparable, 2) time-space
%separable, 3) time-space separable with two spatial rfs for on/off
%respectively. Returns log likelihood as well as the gradient for each
%parameters.
    %% compute log likelihood
    log_likelihood = 0;
    if size(glm_weights,1) == size(stim,2)
        how = 0; %time space nonseparable
        weight = glm_weights;
    elseif size(glm_weights,1) == temporal_len+grid_size^2
        % time-space separable rf
        how = 1; %time space separable
        temporal_rf = glm_weights(1:temporal_len);
        spatial_rf = glm_weights(temporal_len+1:end);
        weight = make_3d_rf(temporal_rf, spatial_rf);
    elseif size(glm_weights,1) == temporal_len+2*(grid_size^2)
        how = 2; %time space separable on off separable
        temporal_rf = glm_weights(1:temporal_len);
        spatial_rf_on = glm_weights(temporal_len+1:temporal_len+grid_size^2);
        spatial_rf_off = glm_weights(temporal_len+grid_size^2+1:end);
        weight_on = make_3d_rf(temporal_rf, spatial_rf_on);
        weight_off = make_3d_rf(temporal_rf, spatial_rf_off);
    end

    if how == 0 || how ==1
       grad_stim = zeros(size(stim));
    elseif how == 2
       grad_stim_on = zeros(size(stim));
       grad_stim_off = zeros(size(stim));
       stim_on = stim;
       stim_on(stim_on<0) = 0;
       stim_off = stim;
       stim_off(stim_off>0) = 0;
    end
    for i=1:size(stim,1)
        if how == 0 || how == 1
            log_likelihood = log_likelihood + spike_train(i).*log(td*exp(stim(i,:)*weight)) ...
                - td*exp(stim(i,:)*weight) - log(factorial(spike_train(i)));
            grad_stim(i,:) = (spike_train(i) - td*exp(stim(i,:)*weight)).*stim(i,:);
        elseif how == 2
            log_likelihood = log_likelihood + spike_train(i).*log(td*exp(stim_on(i,:)*weight_on + stim_off(i,:)*weight_off)) ...
                - td*exp(stim_on(i,:)*weight_on + stim_off(i,:)*weight_off) - log(factorial(spike_train(i)));
            grad_scale = spike_train(i) - td*exp(stim_on(i,:)*weight_on + stim_off(i,:)*weight_off);
            grad_stim_on(i,:) = grad_scale.*stim_on(i,:);
            grad_stim_off(i,:) = grad_scale.*stim_off(i,:);
        end
    end
    neg_log_likelihood = -log_likelihood;
    negLL = sprintf('Neg Log Likelihood: %0.3f', neg_log_likelihood);
```

```
53      disp(negLL);
54      %% compute gradient w.r.t each weights (theta) for fmin solver
55      if how == 0
56          dwdth = eye(length(weights));
57      elseif how == 1
58          theta_len = temporal_len+grid_size^2;
59          dwdth = compute_dwdth_separable(theta_len, temporal_rf, spatial_rf);
60      elseif how == 2
61          theta_len = temporal_len+grid_size^2;
62          dwdth_on = compute_dwdth_separable(theta_len, temporal_rf, spatial_rf_on);
63          dwdth_off = compute_dwdth_separable(theta_len, temporal_rf, spatial_rf_off);
64          %padding zeros for actual dwdth computation
65          dwdth_on_pad = zeros(temporal_len*grid_size^2, theta_len+grid_size^2);
66          dwdth_off_pad = zeros(temporal_len*grid_size^2, theta_len+grid_size^2);
67          dwdth_on_pad(:,1:theta_len) = dwdth_on;
68          dwdth_off_pad(:,1:temporal_len) = dwdth_off(:,1:temporal_len);
69          dwdth_off_pad(:,temporal_len+grid_size^2+1:end) = dwdth_off(:,temporal_len+1:
                end);
70      end
71
72      if how == 0 || how == 1
73          dL = grad_stim*dwdth;
74          dL = -sum(dL,1)';
75      elseif how == 2
76          dL = grad_stim_on*dwdth_on_pad + grad_stim_off*dwdth_off_pad;
77          dL = -sum(dL,1);
78      end
79      grad_norm = sprintf('gradient norm length: %0.3f', norm(dL));
80      disp(grad_norm);
81  end
```

The inputs to the function are:

1) **stim**: The design matrix

2) **spike_train**: Corresponding spike counts per row

3) **glm_weight**: Initial values of θ. Can usually be taken / derived from the spike-triggered-average.

4) **td**: Time delta, or Δ defined in the equation in section 2.

5) **temporal_len**: Length of the temporal receptive field. Has to be consistent with the size of **glm_weight**

6) **grid_size**: Length of a single dimension of the visual stimulus. (Only written for when the two spatial dimensions are the same)

Line by line documentation:

line 9 - 26:

This creates the **weight** variable, which is what we wrote as $\vec{w}$ above. Thus, the weight is a flattened three-dimensional array of dimensions (**grid_size** x **grid_size** x **temporal_len**). Notice for on-off separated case, we have two **weight** variables, one for each of on and off.

For time-space separable case, I am using the **make_3d_rf** function, which is defined as below:

```
function weight = make_3d_rf(temporal_rf, spatial_rf)
    temporal_len = size(temporal_rf,1);
    spatial_len = size(spatial_rf,1);
    weight = repmat(spatial_rf, [temporal_len, 1]);
    weight = weight.*repelem(temporal_rf, spatial_len);
end
```

This function just multiplies the 2-dimensional spatial receptive field matrix with the corresponding values in the temporal receptive field for each temporal position to create the three-dimensional weight matrix.

Line by line documentation:

line 4: First, just replicate the **spatial_rf** in the temporal direction to initialize the 3d weight matrix.

line 5: Repelem function creates an array of the same size, by taking the temporal rf and repeating the values, so that a slice in the temporal dimension would be a square array with the same value.
Then, by taking an element-wise product, the **weight** variable is created.

Going back to the main function:

Let's go back to the partial derivative equation we derived above:

$$\frac{\partial L}{\partial \theta_i} = \sum_{t=1}^{T} [(y_t - \Delta exp(\vec{s}_t \cdot \vec{w})) * \vec{s}_t \cdot \frac{\partial \vec{w}}{\partial \theta_i}] \tag{5}$$

One can see that the equation can be divided into two parts:
1)

$$(y_t - \Delta exp(\vec{s}_t \cdot \vec{w})) * \vec{s}_t \tag{6}$$

2)

$$\frac{\partial \vec{w}}{\partial \theta_i} \tag{7}$$

Let's look at the 1) first. The component within the parenthesis is a scalar, while $\vec{s}_t$ is a row vector.

2) is a column vector, and we are taking a dot product.

Although there is a summation enclosing the whole, we can translate 1) and 2) into a matrix and a matrix multiplication and worry about the summation later.

To be more specific, we can create:

1) a matrix of the same size with the **stim**, where each row is scaled by the scaling factor $(y_t - \Delta exp(\vec{s}_t \cdot \vec{w}))$.

2) another matrix of size (length of **weight** x length of θ), where the ith columns of the matrix is $\frac{\partial \vec{w}}{\partial \theta_i}$.
You can see that multiplying the two matrix will create another matrix of the size (length of rows of **stim** x length of θ), where each element saves the resulting value of the dot product between a row in scaled stimulus, and $\frac{\partial \vec{w}}{\partial \theta_i}$.
Then, we can sum along the first dimension, to get each value of $\frac{\partial L}{\partial \theta_i}$.

line 28 - 37: Initializes a matrix called **grad_stim**, which corresponds to the matrix number 1 stated above. Each row will be filled later with a scaled stimulus row.

line 38 - 56: Iterates through each row of the stimulus to compute the log-likelihood as well as the intermediate matrices needed for computing the gradient.

line 55 - 70: Computes the second matrix stated above to compute the gradient.

For time-space nonseparable case, the θ is just the same as the **weight** vector, thus making the matrix just an identity matrix (line 56).

However, for time-space separable case, each value of the weight vector is a value in spatial and temporal rf multiplied. This is a little more tricky to compute. The function **compute_dwdth_separable** serves this functionality.

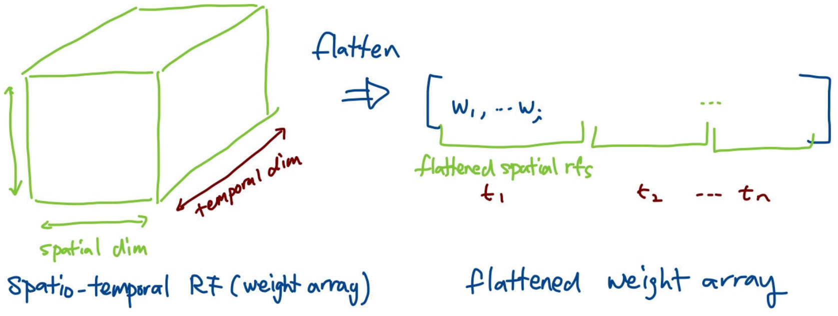
```
function dwdth = compute_dwdth_separable(theta_len, temporal_rf, spatial_rf)
    temporal_len = length(temporal_rf);
    spatial_len = length(spatial_rf);
    weight_len = temporal_len*spatial_len;
    dwdth = zeros(weight_len, theta_len);

    for i = 1:theta_len
        if i<=temporal_len
            dwdthi = zeros(weight_len,1);
            dwdthi((i-1)*spatial_len+1:i*spatial_len) = spatial_rf;
            dwdth(:,i) = dwdthi;
        else
            dwdthi = zeros(size(spatial_rf));
            dwdthi(i-temporal_len) = 1;
            dwdthi = repmat(dwdthi, [temporal_len, 1]);
            dwdthi = dwdthi.*repelem(temporal_rf, spatial_len);
            dwdth(:,i) = dwdthi;
        end
    end
end
```

In the time-space separable case, the θ vector is a concatenation of the temporal rf and the spatial rf in the order mentioned.

Thus, dealing with the temporal θ element and the spatial θ element needs to be taken care of separately.

Refer to the images below for detailed explanation.



When θ_i = temporal rf, $\theta_i = t_i$

$$\frac{\partial \vec{w}}{\partial \theta_i} = \frac{\partial}{\partial t_i}\Bigg(\Big[\underbrace{\underbrace{\qquad}_{t_1}\ \underbrace{\qquad}_{t_2}\ \cdots}_{\text{becomes all zeros since no } t_i \text{ component}}\ \underbrace{\underbrace{\overset{w_i}{\qquad}}_{t_i}}_{w_i = w_{sp} \times t_i}\ \cdots\ \underbrace{\qquad}_{\text{zeros}}\Big]\Bigg)$$

$$= \Big[\underbrace{\text{zeros}}_{t_1} \ ----\ \underbrace{w_{sp}\cdots}_{t_i}\ ---\ \underbrace{\text{zeros}}_{t_n}\Big]$$

t_i ↓ Index: $(i-1) \times$ spatial_rf_len$+1 \sim i \times$ spatial_rf_len when θ_i = spatial rf , $\theta_i = s_{jk}$

Spatial RF

$$\begin{bmatrix} s_{11} & & & \\ s_{21} & \ddots & & \\ \vdots & & s_{jk} & \\ \vdots & & & \ddots \\ & & & & s_{nn} \end{bmatrix}$$

$$\frac{\partial \tilde{w}}{\partial \theta_i} = \frac{\partial}{\partial s_{jk}}\left(\left[\underbrace{t_1 \cdot s_{11} \cdots \underbrace{t_1 \cdot s_{jk}} \cdots}\ \ \underbrace{t_2 \cdot s_{11} \cdots \underbrace{t_2 \cdot s_{jk}}}\ \ \cdots \right]\right)$$

only these elements survive.

$$= \left[\underbrace{0\,0\,0 \cdots t_1 \cdots 0\,0}\ \ \underbrace{0\,0 \cdots \boxed{t_2} \cdots}\ \ \underbrace{0\,0\ t_3\ 0\,0}\ \cdots\ \underbrace{0\,0\ t_n\ 0\,0}\right]$$

Index: $(2-1) \times$ spatial_rf_len $+ (j \times$ grid_size$) + k$

Going back to the main function:

line 72 - 80: Compute the intermediate gradient matrix by multiplying **grad_stim** and **dwdth**. Then, in ***line 74, 77***, we sum the matrix along the first dimension to get the resulting gradient vector.