Notes on generalized linear models for neuroscience

John J. Vastola

February 19, 2024

Abstract

These notes introduce generalized linear models (GLMs), a family of simple statistical models that in some sense generalizes the idea of fitting a line to data. First, we cover how to define them and how to fit model parameters given data. Then we discuss the linear-nonlinear-Poisson model, the GLM most often used in neuroscience, and a simple application to data from the medial entorhinal cortex.

1 Resources

For more information about GLMs, see the following resources:

- Murphy, Book 1, Ch. 12: good coverage of basic GLM theory
- 2016 SFN tutorial on GLMs by Jesse Kaminsky and Jonathan Pillow
- Neuromatch Academy GLM tutorial, part 1
- Neuromatch Academy GLM tutorial, part 2

For research code that fits GLMs to neural data, see:

- GLM_Tensorflow_2: repo with GLM tutorials and Python code for fitting neural data
- ln-model-of-mec-neurons: Giocomo lab Matlab code for fitting MEC neural data
- spline-lnp-model: Giocomo lab Matlab code for fitting MEC neural data, advanced
- Pillow lab GLM code

2 Introduction

In neuroscience, statistical models are essential for making sense of the complex link between neural activity and things in the world (e.g., animal behavior). Among the most useful statistical models are linear ones, because (i) they are simple, and (ii) they are interpretable. Generalized linear models (GLMs) [1, 2], not to be confused with *general* linear models, are a slight complexification of the idea of linear models, and are widely applied in neuroscience. For informative discussion of the high-level idea and some early history, see McCulloch [3].

Early applications of GLMs to neuroscience used linear-nonlinear-Poisson models to study neural data from the macaque primary motor cortex [4] and retina [5]. More recent work has applied them to understand neural dynamics related to decision-making in the macaque lateral intraparietal area (LIP) [6], to characterize tuning and quantify mixed selectivity in mouse medial entorhinal cortex (MEC) [7, 8], and to characterize the posterior-cortex-wide encoding of movement- and decision-making-related variables during dynamic navigation tasks [9, 10]. This list of applications is of course far from exhaustive, but at least illustrates the power and flexibility of GLMs.

In these notes, our goal is to cover the basic theory and a simple application to real neural data from mouse MEC.

3 Review of regression

Since GLMs generalize standard regression approaches, it is helpful to start by reviewing the details of those approaches. We do this for three common types of regression below.

3.1 Linear regression

Linear regression formalizes the idea that an output $y \in \mathbb{R}$ relates to an input $\boldsymbol{x} \in \mathbb{R}^D$ in a 'close-to-linear' fashion[1]. Since real data involves noise, we do not assume a precisely linear relationship; instead, we assume something like

$$y = \boldsymbol{w}^T \boldsymbol{x} + w_0 + r \tag{1}$$

where r is a noise variable. One particularly simple choice assumes r is normally distributed with mean zero, i.e., $r \sim \mathcal{N}(0, \sigma^2)$. The generative model that links inputs and outputs is

$$p(y|\boldsymbol{x}, \boldsymbol{w}, \sigma^2) = \mathcal{N}(y; \boldsymbol{w}^T \boldsymbol{x} + w_0, \sigma^2) \ . \tag{2}$$

Note that this means

$$\mathbb{E}[y|\boldsymbol{x}] = \boldsymbol{w}^T \boldsymbol{x} + w_0 \ , \tag{3}$$

[1] We will assume one-dimensional outputs here for simplicity; the general case is not substantially different.

i.e., that the mean output is linear in the input. The corresponding log-likelihood of a set of N data points $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$ is

$$\log p = -\sum_{n=1}^{N} \frac{(y_n - \boldsymbol{w}^T \boldsymbol{x}_n - w_0)^2}{2\sigma^2} \ . \tag{4}$$

We are interested in estimating the weights $\boldsymbol{w} = (w_1, ..., w_D)^T$ and w_0 given our data, and can do this by maximizing the log-likelihood of the data with respect to the weights. The relevant derivatives are

$$\begin{aligned} \frac{\partial \log p}{\partial \boldsymbol{w}} &= \sum_n \frac{(y_n - \boldsymbol{w}^T \boldsymbol{x}_n - w_0)\boldsymbol{x}_n}{\sigma^2} \\ \frac{\partial \log p}{\partial w_0} &= \sum_n \frac{(y_n - \boldsymbol{w}^T \boldsymbol{x}_n - w_0)}{\sigma^2} \ . \end{aligned} \tag{5}$$

Because the above equations are linear, we can analytically solve for the optimal weights $\boldsymbol{w}^*$ and w_0^*; for other kinds of regression, this will usually not be possible, forcing us to optimize weights using something like gradient descent.

A useful way of rewriting these gradients is in terms of a $N \times (D+1)$ *design matrix* $\boldsymbol{X}$, which is for this model defined via

$$X_{ni} = \begin{cases} 1 & \text{for } i = 0 \\ x_{ni} & \text{for } i = 1, ..., D \end{cases} \ . \tag{6}$$

The design matrix is useful because it describes our input samples in terms of the 'features' (i.e., the variables x_i for $i = 1, ..., D$ and the constant 1) our model depends on. Also define the vectors $\boldsymbol{Y} := (y_1, ..., y_N)^T \in \mathbb{R}^N$ and $\boldsymbol{W} := (w_0, w_1, ..., w_D)^T \in \mathbb{R}^{D+1}$. The gradients can now be written as

$$\frac{\partial \log p}{\partial \boldsymbol{W}} = \frac{1}{\sigma^2} \boldsymbol{X}^T \left[\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{W} \right] \ , \tag{7}$$

and the optimal weights are

$$\boldsymbol{W}^* = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{Y} \ . \tag{8}$$

A final useful thing to point out is that the gradient of the log-likelihood has a particularly nice form that resembles a two-factor learning rule: each component is a product of (i) the difference between the true and predicted output, and (ii) an input feature. Explicitly,

$$\begin{aligned} \frac{\partial \log p}{\partial \boldsymbol{w}} &= \sum_n \frac{[y_n - \hat{y}(\boldsymbol{x}_n; \boldsymbol{w}, w_0)]\boldsymbol{x}_n}{\sigma^2} \\ \frac{\partial \log p}{\partial w_0} &= \sum_n \frac{[y_n - \hat{y}(\boldsymbol{x}_n; \boldsymbol{w}, w_0)]}{\sigma^2} \ . \end{aligned} \tag{9}$$

3.2 Logistic regression

Logistic regression relates binary outputs $y \in \{0, 1\}$ to real-valued inputs $\boldsymbol{x} \in \mathbb{R}^D$. It is perhaps the simplest canonical model of *classification*; given examples $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$, we are interested in classifying an unseen input as either zero or one.

From a generative modeling point of view, we imagine that there is a threshold (a line in $\mathbb{R}^D$), and that the probability of classifying an example as zero or one depends somewhat on its distance to the threshold. Examples clearly on one or the other side get classified as zero or one, while examples near the threshold can be classified as either. This is formalized by writing

$$p(y|\boldsymbol{x}, \boldsymbol{w}, w_0) = \text{Binom}(y; \sigma(\boldsymbol{w}^T\boldsymbol{x} + w_0)) = \sigma(\boldsymbol{w}^T\boldsymbol{x} + w_0)^y \left[1 - \sigma(\boldsymbol{w}^T\boldsymbol{x} + w_0)\right]^{1-y} \tag{10}$$

where $\sigma(\cdot)$ denotes the sigmoid function

$$\sigma(z) = \frac{1}{1 + e^{-z}} \ . \tag{11}$$

Note that the mean output is no longer linear in the input, but relates to it according to

$$\mathbb{E}[y|\boldsymbol{x}] = \sigma(\boldsymbol{w}^T\boldsymbol{x} + w_0) \in [0, 1] \ . \tag{12}$$

This is quite similar to what we found for linear regression (Eq. 3), except for the $\sigma(\cdot)$ function applied to the linear term. The sigmoid function is an example of an 'activation' function, and its inverse is an example of a 'link' function.

The corresponding log-likelihood of a set of data is

$$\log p = \sum_n y_n \log \sigma(\boldsymbol{w}^T\boldsymbol{x}_n + w_0) + (1 - y_n) \log \left[1 - \sigma(\boldsymbol{w}^T\boldsymbol{x}_n + w_0)\right] \ . \tag{13}$$

Using the fact that $\sigma' = \sigma(1 - \sigma)$, the gradient of $\log p$ with respect to the weights is

$$\begin{aligned}
\frac{\partial \log p}{\partial \boldsymbol{w}} &= \sum_n y_n \frac{\sigma'(\boldsymbol{w}^T\boldsymbol{x}_n + w_0)\boldsymbol{x}_n}{\sigma(\boldsymbol{w}^T\boldsymbol{x}_n + w_0)} - (1 - y_n)\frac{\sigma'(\boldsymbol{w}^T\boldsymbol{x}_n + w_0)\boldsymbol{x}_n}{1 - \sigma(\boldsymbol{w}^T\boldsymbol{x}_n + w_0)} \\
&= \sum_n \left\{y_n \left[1 - \sigma(\boldsymbol{w}^T\boldsymbol{x}_n + w_0)\right] - (1 - y_n)\sigma(\boldsymbol{w}^T\boldsymbol{x}_n + w_0)\right\} \boldsymbol{x}_n \\
&= \sum_n \left[y_n - \sigma(\boldsymbol{w}^T\boldsymbol{x}_n + w_0)\right] \boldsymbol{x}_n \\
\frac{\partial \log p}{\partial w_0} &= \sum_n y_n - \sigma(\boldsymbol{w}^T\boldsymbol{x}_n + w_0) \ .
\end{aligned} \tag{14}$$

If we view the model's mean output as a prediction of the label, we can write

$$\begin{aligned}
\frac{\partial \log p}{\partial \boldsymbol{w}} &= \sum_n \left[y_n - \hat{y}(\boldsymbol{x}_n; \boldsymbol{w}, w_0)\right] \boldsymbol{x}_n \\
\frac{\partial \log p}{\partial w_0} &= \sum_n y_n - \hat{y}(\boldsymbol{x}_n; \boldsymbol{w}, w_0) \ .
\end{aligned} \tag{15}$$

3.3 Poisson regression

Poisson regression relates real-valued input $\boldsymbol{x} \in \mathbb{R}^D$ to nonnegative-integer-valued output $y \in \mathbb{N} = \{0, 1, 2, ...\}$. The generative model is

$$p(y|\boldsymbol{x}, \boldsymbol{w}, w_0) = \mathcal{P}(y; \exp\big(\boldsymbol{w}^T\boldsymbol{x} + w_0\big)) = \frac{e^{y(\boldsymbol{w}^T\boldsymbol{x}+w_0)}e^{-\exp\big(\boldsymbol{w}^T\boldsymbol{x}+w_0\big)}}{y!} \tag{16}$$

and the average output conditional on the input is

$$\mathbb{E}[y|\boldsymbol{x}] = e^{\boldsymbol{w}^T\boldsymbol{x}+w_0} \in [0, \infty) \ . \tag{17}$$

As in the case of logistic regression, the mean output is no longer a linear function of the input; instead, it is related to it through a so-called activation function, which in this case is exponential. The log-likelihood of a set of data is

$$\log p = \sum_n y_n(\boldsymbol{w}^T\boldsymbol{x}_n + w_0) - e^{\boldsymbol{w}^T\boldsymbol{x}_n+w_0} \tag{18}$$

where we have dropped the $-\log(y!)$ terms, since they do not depend on the model parameters. The gradient of $\log p$ with respect to the weights is

$$\begin{aligned} \frac{\partial \log p}{\partial \boldsymbol{w}} &= \sum_n \left[y_n - e^{\boldsymbol{w}^T\boldsymbol{x}_n+w_0}\right] \boldsymbol{x}_n \\ \frac{\partial \log p}{\partial w_0} &= \sum_n y_n - e^{\boldsymbol{w}^T\boldsymbol{x}_n+w_0} \ . \end{aligned} \tag{19}$$

Viewing the model's mean output as a prediction, we can write

$$\begin{aligned} \frac{\partial \log p}{\partial \boldsymbol{w}} &= \sum_n \left[y_n - \hat{y}(\boldsymbol{x}_n; \boldsymbol{w}, w_0)\right] \boldsymbol{x}_n \\ \frac{\partial \log p}{\partial w_0} &= \sum_n y_n - \hat{y}(\boldsymbol{x}_n; \boldsymbol{w}, w_0) \ . \end{aligned} \tag{20}$$

4 Generalized linear models

One may have noticed that the three preceding examples had a few things in common. In each case, we relate inputs $\boldsymbol{x} \in \mathbb{R}^D$ to outputs y via a generative model with the property

$$\mathbb{E}[y|\boldsymbol{x}] = f(\boldsymbol{w}^T\boldsymbol{x} + w_0) \tag{21}$$

where $f : \mathbb{R} \to \mathbb{R}$ is an activation function. In the earlier examples, we had:

- $f(z) = z$ (linear regression);
- $f(z) = \sigma(z)$ (logistic regression);
- $f(z) = \exp(z)$ (Poisson regression).

We also derived gradient descent parameter update rules of the form

$$\begin{aligned} \frac{\partial \log p}{\partial \boldsymbol{w}} &= \sum_n \left[y_n - \hat{y}(\boldsymbol{x}_n; \boldsymbol{w}, w_0)\right] \boldsymbol{x}_n \\ \frac{\partial \log p}{\partial w_0} &= \sum_n y_n - \hat{y}(\boldsymbol{x}_n; \boldsymbol{w}, w_0) \end{aligned} \tag{22}$$

where $\hat{y}(\boldsymbol{x}_n; \boldsymbol{w}, w_0) := \mathbb{E}[y|\boldsymbol{x}_n]$. The idea of a generalized linear model (GLM) is to consider the most general class of statistical models for which we have analogously nice behavior. To paraphrase McCulloch [3], defining a GLM involves making three decisions:

1. According to what probability distribution is the data generated?
2. What activation function relates a linear combination of inputs to the mean output?
3. What states do we use as model input, and what state representation do we use?

The first choice is restricted by the format of the data (e.g., it makes more sense to model spike count data as discrete than continuous). The mathematical framework associated with GLMs offers a specific recommendation for the second choice given the first choice, as we will see shortly. The third choice is highly domain-dependent, and it is a major open problem to decide good state representations for capturing (for example) what neurons 'care' most about.

4.1 Definition of GLMs

We will follow Murphy's [11] treatment of GLMs (see Murphy, Book, 1, Ch. 12, pg. 413).

Generative model. We need a generative model. Consider an (overdispersed) exponential family distribution

$$p(y|\eta, \sigma^2) = \exp\left\{\frac{y\eta - A(\eta)}{\sigma^2} + \log h(y, \sigma^2)\right\} \tag{23}$$

where η is the natural parameter, h is the base measure, A is the log-normalizer, and $\sigma^2 > 0$ is the dispersion parameter. We could have considered something much more general (e.g., allowing y to be more than one-dimensional, and/or having nontrivial sufficient statistics), but since the GLMs used in practice tend to be of this kind, we will avoid extra complexity.

A nice property of the log-normalizer A (essentially, the constant that normalizes the probability distribution) is that, since it is defined to be

$$A(\eta) := \sigma^2 \log\left\{\int h(y, \sigma^2) e^{\frac{y\eta}{\sigma^2}}\, dy\right\}\ , \tag{24}$$

derivatives of A with respect to η correspond to cumulants of y. For example:

$$\begin{aligned} \frac{\partial A}{\partial \eta} &= \mathbb{E}[y|\eta] \\ \frac{\partial^2 A}{\partial \eta^2} &= \frac{1}{\sigma^2}\mathrm{var}[y|\eta]\ . \end{aligned} \tag{25}$$

Each of the three generative models we considered earlier is a special case of this one. In particular, using μ to denote the mean of each distribution:

- Linear regression: $\eta = \mu$, $A(\eta) = \frac{\mu^2}{2}$, $h = \exp\left(-\frac{y^2}{2\sigma^2}\right)/\sqrt{2\pi\sigma^2}$, $\sigma^2 = \sigma^2$
- Logistic regression: $\eta = \log(\mu/(1-\mu))$, $A(\eta) = -\log(1-\mu)$, $h = 1$, $\sigma^2 = 1$
- Poisson regression: $\eta = \log\mu$, $A(\eta) = \mu$, $h = 1/y!$, $\sigma^2 = 1$

Activation function. We must next decide how to relate inputs $\boldsymbol{x}$ to the generative model. The GLM idea is to do this by imposing the relationship

$$\mathbb{E}[y|\boldsymbol{x}] = f(\boldsymbol{w}^T\boldsymbol{x} + w_0) \tag{26}$$

for some activation function f and weights $\boldsymbol{w}$ and w_0. Imposing this relationship relates $\boldsymbol{x}$ to μ, which determines η, which hence precisely determines the form of the generative model. (Terminology note: we also speak of the 'link function', which is defined as f^{-1}.)

Although we in principle are free to choose f however we like (and this is sometimes done in practice), there is a special theoretically-motivated choice with nice properties. Let

$\gamma : \mathbb{R} \to \mathbb{R}$ be the (assumed invertible) function that maps the natural parameter η to the mean μ, i.e., $\gamma(\eta) = \mu$. We have the relationships

$$\eta \underset{\gamma^{-1}}{\overset{\gamma}{\rightleftarrows}} \mu \underset{f}{\overset{f^{-1}}{\rightleftarrows}} \boldsymbol{w}^T \boldsymbol{x} + w_0 \tag{27}$$

between η, μ, and the input. The canonical choice assumes that $f = \gamma$, which means that the natural parameter depends linearly on the input, i.e.,

$$\eta = \boldsymbol{w}^T \boldsymbol{x} + w_0 \ . \tag{28}$$

Given this choice, the generative model becomes

$$p(y|\eta, \sigma^2) = \exp\left\{ \frac{y(\boldsymbol{w}^T \boldsymbol{x} + w_0) - A(\eta)}{\sigma^2} + \log h(y, \sigma^2) \right\} \ , \tag{29}$$

which implies that the gradients of the log-likelihood take a particularly simple form:

$$\begin{aligned} \frac{\partial \log p}{\partial \boldsymbol{w}} &= \frac{1}{\sigma^2} \sum_n \left[y_n - \frac{\partial A}{\partial \eta} \right] \boldsymbol{x}_n \\ &= \frac{1}{\sigma^2} \sum_n \left[y_n - \hat{y}(\boldsymbol{x}_n; \boldsymbol{w}, w_0) \right] \boldsymbol{x}_n \\ \frac{\partial \log p}{\partial w_0} &= \frac{1}{\sigma^2} \sum_n \left[y_n - \hat{y}(\boldsymbol{x}_n; \boldsymbol{w}, w_0) \right] \ . \end{aligned} \tag{30}$$

In all three previously described kinds of regression, the canonical activation function was used. If a noncanonical link function is used instead, the gradients have a slightly less nice form, but this is not necessarily a huge problem in the era of automatic differentiation packages.

State representation. It is important to note that the inputs $\boldsymbol{x}$ are whatever we like, and reflect the variables we think best predict the output y. The inputs in particular are not necessary raw behavioral signals we record, like an animal's instantaneous velocity or heading, but can rather be arbitrarily complicated functions of these variables. More complex choices allow us to fit more complicated data, but can leave us vulnerable to overfitting-related issues if we have too many parameters and too few data points.

5 Linear-nonlinear-Poisson regression in practice

In practice, neuroscience applications involve a fairly simple kind of GLM: the linear-nonlinear-Poisson model (Fig. 1). The biophysical picture implied by this statistical model is that each neuron in a population fires independently of the others with Poisson-like statistics, and is tuned to particular behavioral (e.g., instantaneous movement) or cognitive (e.g., evidence accumulated for a particular choice) variables. Including feedback within the GLM, i.e., including the previous y as one of the features that determines the next y, permits GLMs to capture more diverse neural dynamics, like bursting and refractory periods; see Weber and Pillow [12] for some discussion of this. Below, we will review the simplest form of the Poisson model, which does not include a post-spike filter, for simplicity.

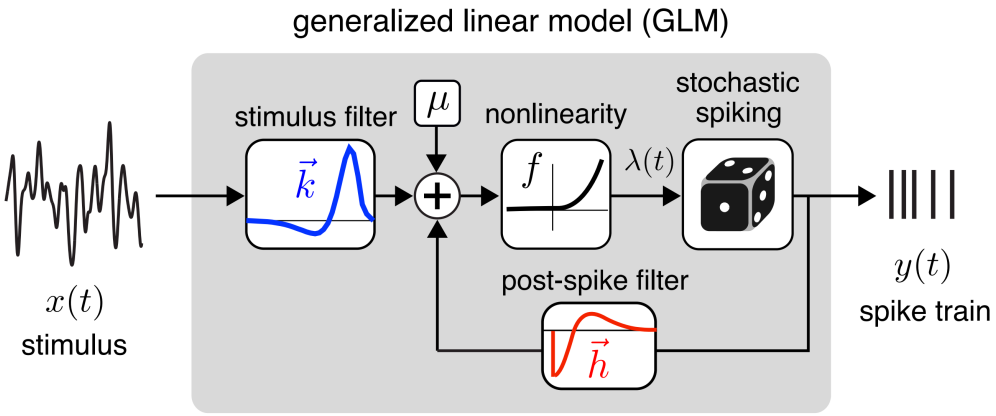

Figure 1: GLM schematic taken from Weber and Pillow 2017 [12]. A user-defined stimulus is linearly transformed, and then passed through a nonlinear activation function f, to determine the mean of a Poisson distribution that determines stochastic spike generation. A post-spike filter can be included to capture more diverse spiking dynamics.

5.1 Generative model and objective function

The generative model and objective function are nearly the same as the ones we discussed earlier (Sec. 3.3). We collect the relevant information below for the reader's convenience:

$$
\begin{aligned}
p(y|\boldsymbol{x}, \boldsymbol{w}, w_0) = \mathcal{P}(y; \exp\big(\boldsymbol{w}^T\boldsymbol{x} + w_0\big)\Delta t) &= \frac{e^{y(\boldsymbol{w}^T\boldsymbol{x}+w_0+\log(\Delta t))}e^{-\exp\big(\boldsymbol{w}^T\boldsymbol{x}+w_0\big)\Delta t}}{y!} \\
\mathbb{E}[y|\boldsymbol{x}] &= \hat{y}(\boldsymbol{x}_n; \boldsymbol{w}, w_0) = \exp\big(\boldsymbol{w}^T\boldsymbol{x} + w_0\big)\Delta t \\
\log p &= \sum_n y_n(\boldsymbol{w}^T\boldsymbol{x}_n + w_0) - e^{\boldsymbol{w}^T\boldsymbol{x}+w_0}\Delta t \\
\frac{\partial \log p}{\partial \boldsymbol{w}} &= \sum_n \left[y_n - \hat{y}(\boldsymbol{x}_n; \boldsymbol{w}, w_0)\right]\boldsymbol{x}_n \\
\frac{\partial \log p}{\partial w_0} &= \sum_n y_n - \hat{y}(\boldsymbol{x}_n; \boldsymbol{w}, w_0)
\end{aligned}
\tag{31}
$$

Here we have assumed a canonical (exponential) link function for simplicity. The Δt factor requires some explanation. If we are fitting spiking activity (or something morally equivalent, like deconvolved calcium activity), we usually use time-binned activity, and in the simplest case treat the number of spikes in each time bin as independent samples from the same distribution. The Δt factor takes care of the fact that how many spikes we observe depends on the size of our time bins, and ensures that we can interpret the $f(\cdot)$ part of the model as characterizing a firing rate. This also allows us to interpret the weights as specifying tuning curves with respect to different variables, as we will see in the next section.

5.2 Regularization

We often work in an overparameterized, data-limited regime, which requires us to use regularization in our parameter fitting procedure in order to avoid overfitting-related issues. Three types of regularization are common and useful here: (i) encouraging parameter smoothness, (ii) penalizing parameter magnitude, and (iii) encouraging parameter sparsity.

We will say more about the first kind in the next section, but this kind of penalty is what allows us to (for example) recover smooth tuning curves when analyzing mouse MEC data. Penalizing parameter magnitude is empirically known to help generalization, and encouraging sparsity emphasizes weights that correspond to features a neuron is highly responsive to, and eliminates weights that it barely depends on (which may be a statistical artifact).

The standard way to incorporate the latter two kinds of regularization is to include a mix of L1 and L2 regularization terms on the weights in the objective. The second two types of regularization can also be combined, for example by using a (group-) lasso penalty. See Tseng and Chettih et al. [10] for details.

5.3 Quantifying goodness-of-fit

In a linear regression setting, a canonical way of assessing model fit is using 'variance explained', a quantity which relies on a particular decomposition of the log-likelihood. Although there are problems with using the exact same measure to assess the goodness-of-fit of a Poisson regression model, an analogous quantity can be defined.

The idea is to compare the log-likelihood of a given model fit to two extremes. One extreme is the **saturated model**, which involves a separate parameter for each data point, and hence fits the data perfectly. The other extreme is a **null model**, which only fits the mean of the data (equivalently: only the w_0 term is retained, and all $\boldsymbol{x}$-dependence is removed). The saturated model fits perfectly, and null model fits the worst of all possible models. If our inputs $\boldsymbol{x}$ can actually predict the outputs y, then a fitted model should have a log-likelihood somewhere in-between—and ideally closer to the saturated model. This would indicate that one does *not* actually need a separate parameter for every data point in order to fit the data well.

Let us be more precise. Let ℓ_S denote the log-likelihood of the saturated model, ℓ_0 denote the log-likelihood of the null (intercept-only) model, ℓ_M denote the log-likelihood of a nontrivial regression model that includes some kind of $\boldsymbol{x}$-dependence, and $\langle y \rangle := \sum_n y_n / N$ denote the mean of the output samples. Define the **fraction deviance explained** as[2]

$$R^2_{GLM} := \frac{D_{model}}{D_{tot}} \geq 0 \ , \tag{32}$$

where the explained deviance D_{model} is defined via

$$\begin{aligned} D_{model} &= 2(\ell_M - \ell_0) \\ &= 2 \sum_n \left[\ y_n \log(\mu_n / \langle y \rangle) - (\mu_n - \langle y \rangle) \ \right] \\ &= 2 \sum_n \Big[\ y_n \left(\boldsymbol{w}^T \boldsymbol{x}_n + w_0 - \log \langle y \rangle \right) - (e^{\boldsymbol{w}^T \boldsymbol{x}_n + w_0} - \langle y \rangle) \ \Big] \end{aligned} \tag{33}$$

and the total deviance D_{tot} is defined via

$$D_{tot} := 2(\ell_S - \ell_0) = 2 \sum_n \left[\ y_n \log(y_n / \langle y \rangle) - (y_n - \langle y \rangle) \ \right] \ . \tag{34}$$

See the methods section of Tseng and Chettih et al. [10] for some comments about using Poisson deviance as a measure of goodness-of-fit. In practice, a 'good' fit may have an R^2_{GLM} value of larger than 0.2, but not necessarily close to 1. In other words, GLMs are competitive and useful models of neural activity, but not the final word on how the brain works.

As usual, it makes the most sense to assess goodness-of-fit *not* on the training data, but held-out data. For example, it is reasonable to use 80% of data for training, and the remaining 20% for assessing fit quality.

[2] This quantity is often, but not always, less than 1 when evaluated on held-out data.

6 Example application to MEC data

In this last section, we will use the simple Poisson GLM approach described in the previous section to fit neural data from mouse MEC. We will use data from Mallory and Hardcastle et al. 2021 [8], which is publicly available and can be found here (dataset 1). The dataset includes $n = 179$ neurons collected from the MEC of 6 freely moving mice. These mice performed a foraging task in small rectangular arenas. For more details, see the methods section of the paper.

State representation. We consider a simple GLM motivated by the one considered in the earlier work of Hardcastle et al. 2017 [7]. We assume that neurons respond to some combination of (i) current 2D position, (ii) current heading direction, and (iii) current body speed. Additional behavioral variables were also tracked, and you can play around with incorporating those too if you like.

For simplicity, we consider one-hot encodings of binned position, heading, and body speed. This means that the domain of these variables is discretized (e.g., the $[0, 2\pi)$ range of heading is discretized into some number of bins), and at a given moment of time, a feature corresponding to each bin either takes value 1 (e.g., if the animal is facing in that direction) or zero. Each bin gets a separate weight that must be learned, and only a small subset of bins contribute to predicting neural activity at any given time.

Concretely, we have four types of features: $\boldsymbol{x}_P$, $\boldsymbol{x}_H$, $\boldsymbol{x}_S$, and a constant feature 1. We discretize the area into $N_x \times N_y$ position bins, heading into N_H bins, and speed into N_S bins. The notation is a bit cumbersome, but let x' denote the binned x position of the animal, y' denote the binned y position of the animal, h' denote the binned heading, and s' denoted the binned speed. We have

$$\begin{aligned} x_{Pij} &:= \delta_{ix'}\delta_{jy'} \\ x_{Hi} &:= \delta_{ih'} \\ x_{Si} &:= \delta_{is'} \end{aligned} \tag{35}$$

where δ is the Kronecker delta function, which equals one when its indices match and zero otherwise. Our linear combination of predictors is then

$$w_0 + \boldsymbol{w}_P^T \boldsymbol{x}_P + \boldsymbol{w}_H^T \boldsymbol{x}_H + \boldsymbol{w}_S^T \boldsymbol{x}_S = w_0 + \sum_{i,j} w_{Pij} x_{Pij} + \sum_{i=1}^{N_H} w_{Hi} x_{Hi} + \sum_{i=1}^{N_S} w_{Si} x_{Si} \ . \tag{36}$$

In total, this model has $N_x \cdot N_y + N_H + N_S + 1$ parameters that must be learned from data.

Regularization. For simplicity, we will only consider one type of regularization: encouraging parameter smoothness. In particular, we would like the weights that correspond to nearby values of position, heading, and speed to be somewhat related to one another. To

this end, we include the following terms in the objective function:

$$
\begin{aligned}
&- \beta_P \sum_{i,j} \left[\frac{(w_{P,i+1,j} - w_{Pi,j})^2}{4} + \frac{(w_{P,i,j+1} - w_{Pi,j})^2}{4} \right] \\
&- \beta_H \sum_{i} \frac{(w_{H,i+1} - w_{Hi})^2}{2} \\
&- \beta_P \sum_{i} \frac{(w_{S,i+1} - w_{Si})^2}{2} \ .
\end{aligned}
\tag{37}
$$

Note the negative sign, which is necessary since we are maximizing the log-likelihood instead of minimizing it. Note also that this is not the only way to ensure smoothness; an alternative approach would be to use bump-like encodings of variables like position instead of one-hot encodings. Including these regularization terms, the gradient of the objective J (= log-likelihood plus regularization) is

$$
\begin{aligned}
\frac{\partial J}{\partial w_{Pij}} &= \frac{\partial \log p}{\partial w_{Pij}} + \beta_P \left[\frac{(w_{P,i+1,j} + w_{P,i-1,j} + w_{P,i,j+1} + w_{P,i,j-1})}{4} - w_{Pi,j} \right] \\
\frac{\partial J}{\partial w_{Hi}} &= \frac{\partial \log p}{\partial w_{Hi}} + \beta_H \left[\frac{(w_{H,i+1} + w_{H,i-1})}{2} - w_{Hi} \right] \\
\frac{\partial J}{\partial w_{Si}} &= \frac{\partial \log p}{\partial w_{Si}} + \beta_S \left[\frac{(w_{S,i+1} + w_{S,i-1})}{2} - w_{Si} \right] \ .
\end{aligned}
\tag{38}
$$

There is some subtlety associated with handling boundaries here. Since heading is periodic, $N_H + 1$ is treated as 0 and -1 is treated as N_H. Since speed is not periodic, w_{S,N_S+1} and $w_{S,-1}$ are both treated as zero. Position is also not periodic, and we treat it similarly.

See Fig. 2 for an example model fit.

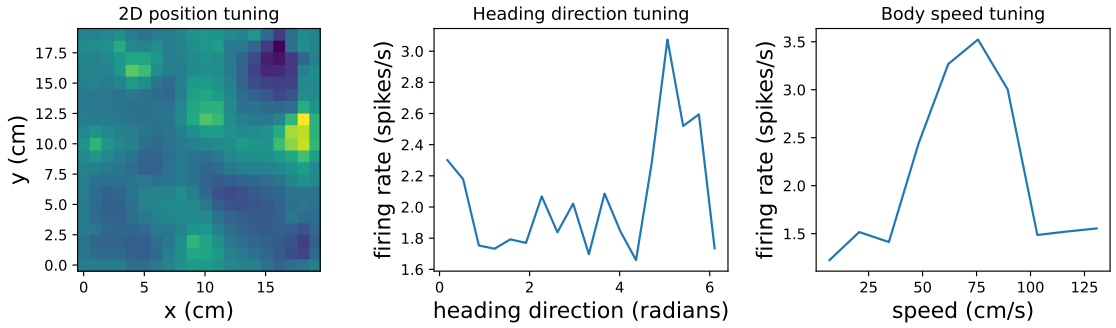

Figure 2: Model fit for one MEC neuron. Tuning curves are obtained by plotting $\hat{y}/\Delta t$ as a function of one type of behavioral variable and ignoring the others.

References

[1] J. A. Nelder and R. W. M. Wedderburn. Generalized linear models. *Journal of the Royal Statistical Society. Series A (General)*, 135(3):370–384, 1972.

[2] P McCullagh and JA Nelder. Generalized linear models. *Generalized Linear Models*, 1989.

[3] Charles E. McCulloch. Generalized linear models. *Journal of the American Statistical Association*, 95(452):1320–1324, 2000.

[4] Wilson Truccolo, Uri T. Eden, Matthew R. Fellows, John P. Donoghue, and Emery N. Brown. A point process framework for relating neural spiking activity to spiking history, neural ensemble, and extrinsic covariate effects. *Journal of Neurophysiology*, 93(2):1074–1089, 2005. PMID: 15356183.

[5] Jonathan W. Pillow, Jonathon Shlens, Liam Paninski, Alexander Sher, Alan M. Litke, E. J. Chichilnisky, and Eero P. Simoncelli. Spatio-temporal correlations and visual signalling in a complete neuronal population. *Nature*, 454(7207):995–999, Aug 2008.

[6] Il Memming Park, Miriam L. R. Meister, Alexander C. Huk, and Jonathan W. Pillow. Encoding and decoding in parietal cortex during sensorimotor decision-making. *Nature Neuroscience*, 17(10):1395–1403, Oct 2014.

[7] Kiah Hardcastle, Niru Maheswaranathan, Surya Ganguli, and Lisa M. Giocomo. A multiplexed, heterogeneous, and adaptive code for navigation in medial entorhinal cortex. *Neuron*, 94(2):375–387.e7, Apr 2017.

[8] Caitlin S. Mallory, Kiah Hardcastle, Malcolm G. Campbell, Alexander Attinger, Isabel I. C. Low, Jennifer L. Raymond, and Lisa M. Giocomo. Mouse entorhinal cortex encodes a diverse repertoire of self-motion signals. *Nature Communications*, 12(1):671, Jan 2021.

[9] Matthias Minderer, Kristen D. Brown, and Christopher D. Harvey. The spatial structure of neural encoding in mouse posterior cortex during navigation. *Neuron*, 102(1):232–248.e11, Apr 2019.

[10] Shih-Yi Tseng, Selmaan N. Chettih, Charlotte Arlt, Roberto Barroso-Luque, and Christopher D. Harvey. Shared and specialized coding across posterior cortical areas for dynamic navigation decisions. *Neuron*, 110(15):2484–2502.e16, Aug 2022.

[11] Kevin P Murphy. *Probabilistic machine learning: an introduction*. MIT press, 2022.

[12] Alison I. Weber and Jonathan W. Pillow. Capturing the dynamical repertoire of single neurons with generalized linear models. *Neural Computation*, 29(12):3260–3289, 2017.