# The generative model

At the most basic level, we assume the response of a voxel is determined by the inner product between its pRF and the stimulus, convolved with the HRF, plus noise:

(1)

where *bi*(*t*) is the bold signal measured in voxel *i* at time *t*, ***s***(*t*) is an *M* x 1 column vector representation of the stimulus (where *M* is the number of pixels)[[1]](#footnote-1), ***Wi*** is a vectorized representation of the receptive field of the *ith* voxel (such that we have a larger matrix **W** = [***W­­****1*, ***W****2*, …, ***W****N*] where *N* is the number of voxels, and so **W** has dimensions *M x N*), *h*(*t*) is the HRF, *c­i* is voxel *i*’s baseline activity, and *εi*(*t*) is the noise in voxel *i* at time *t*, with – that is, the noise is independently normally distributed, and the noise for the *ith* voxel has a variance *σi*2. Note that the formulation in (1) assumes that time is continuous. In practice, we will deal with time in discrete intervals (TRs). So we can rewrite the equation as follows[[2]](#footnote-2):

(2)

where **B** is *N x T* (*T* being the number of time points), **W** is *M* x *N*, **S** is *M* x *T*, **H** is *T* x *T*, **E** is *N* x *T*, **c** is *N* x 1 and **1** is an *N* x *1* column vector of ones[[3]](#footnote-3). Finally, we can also describe our model probabilistically:

(3)

where . This equation gives the probability of the data given some parameters, which, when evaluated as a function of the parameter values, corresponds to the likelihood function of the parameters. Later, when considering regularization techniques for training & testing the model, we will see that these can be thought of as priors imposed on **W** and **S**.

# Training

During training, we want to estimate the matrix **W**, i.e. the pRFs of our voxels, given **S** and **B**. Given the way we have formulated our model, this can be done via a linear regression. However, we will be dealing with a scenario where *M* >> *T* – that is, the number of pixels (and thus the number of coefficients in **W** that need to be estimated for each voxel) is much larger than the number of times that we’re able to measure each voxel’s activity. Therefore, to get an accurate estimate of **W**, we need to regularize our regression. One option, as used by Lee et al., is to use a ridge regression, which penalizes the L2 norm of the estimated coefficients. Intuitively, this means that complex solutions that require many coefficients to be involved (i.e. to have relatively large values) are more ‘expensive’, while simple solutions that require only a small number of coefficients are cheaper. Thus, the ridge regression is biased towards more simple solutions, and this helps us to be more accurate, because we *expect* the solutions to be simple (i.e. we expect voxels to respond strongly to only a small subset of the available pixels). In probabilistic terms, a ridge regression imposes the following prior on **W**:

(4)

Given this prior, the estimate for **W** is given by:

(5)

where (the weights extended with the voxel-wise baselines), ­ (that is, we take the stimulus convolved with the HRF and append a column of 1’s to account for the voxel-wise intercept; thus, the baseline term **c** drops out of the equation), and following Lee et al., , i.e. we take the *M* x *M* identity matrix and extend it with a column and a row of 0’s (this – more precisely the final diagonal element being 0 - ensures that the intercept coefficient is not regularized).

Of course, this requires that we know what value to set *λr* to[[4]](#footnote-4). To determine this, we can use the following cross-validation procedure:

1. Choose a value for *λr*.
2. Given this *λr*, estimate **W** on data from all but one of the pRF runs (this left-out run will serve as the validation run).[[5]](#footnote-5)
3. Using this estimate of **W**, generate predictions for the activations in the validation run.
4. Repeat steps 2-3 until each run has served as the validation run, and thus we have predicted timecourses for all runs.
5. Compute the variance explained (R2) by the predicted timecourses generated in step 3.
6. Repeat steps 1-5 for different values of *λr* (for example in a grid search followed by a pattern search), and finally choose the one that has the highest R2.
7. Estimate **W** on all pRF runs, using this value of *λr*.

Finally, once we have our estimate of **W**, we can use it to compute the noise variances **σ**2:

(6)

In words, we compute the (uncorrected) sample standard deviation of the residuals from each voxel’s predicted timecourse. This is the final step in the training procedure.

If we want to use a graphridge regression rather than a regular ridge regression, not much changes. In a graphridge regression, our expectation about the pRFs is that they aren’t just *simple* (involving only a small subset of all possible pixels), but also *smooth* (the subset of pixels that “participate” in a pRF tend to be located near each other and form a contiguous area, rather than being dotted around the visual field at random). This is captured in the following prior on **W**:

(8)

where **G**, the precision matrix of the Gaussian, is also referred to as a *graphical dependency matrix*. Its diagonal elements are equal to the number of neighbors each pixel has (most pixels have 4, those on the edges have 3 and those in the corners have 2), while the off-diagonal elements are -1 for pixels that are neighbors. Note that this prior reduces to the simple ridge prior if **G** is instead chosen to equal **I+**. In the graphridge regression, this results in a penalty being applied to the differences between the pRF coefficients on neighboring pixels, rather than on the pRF coefficients themselves (so it’s like a higher-order penalty on the structure of the coefficients, instead of on their first-order magnitude).

If we implement graphridge regression, then **W** is estimated as follows:

(9)

where , and *λg* can be selected using the same cross-validation procedure as before.

# Testing

During testing, we want to estimate the stimulus or stimuli that best explains (a) certain pattern(s) of voxel activations, or the difference between two patterns. What we’ve been doing so far is computing average bold patterns across a certain subset of time points, and then doing the decoding on those time points. We have to think a little bit about how such an approach fits with our generative model, because it deals with time a bit differently, and basically ignores or sidesteps the HRF. I think we can do two things here. The first is to use this averaging approach and capture this in some set of formal equations, which I’ll discuss first. The second is to start from the math and develop a slightly different but possibly interesting approach from there. I hope this will become clearer later on.

## The “difference of average bold patterns”-approach

Starting with our familiar approach, let’s try to formalize what we do there. We compute the average bold patterns in the attended and unattended conditions, and then take the difference between those patterns. In terms of equations, that corresponds to:

(10)

where Ais the set of all time points in the attended condition, U is the set of all time points in the unattended condition, denotes the number of elements in a set and **b**(D)is the difference pattern. Then, from equation (2), we get:

Note that the baseline term **c** drops out here due to the subtraction. We can rewrite this equation as follows:

(12)

(11)

In this form, we can recognize as being something like the average stimulus in the attended condition, and as the average stimulus in the unattended condition. However, they aren’t exactly the average stimuli, because they still contain the convolution matrix **H**. Basically, this stems from the fact that we take the raw bold signal and average over it, rather than doing any kind of deconvolution. I don’t think this is a problem though, but we just want to be clear, within our generative model, what it is that we are doing here. For convenience, lets define:

(13)

Filling this into (12), we get:

(14)

which is already looking much friendlier. We’re back to a form that says that the bold pattern is a linear combination of the pixel intensities in the stimulus, plus some noise. To complete the puzzle, all that is left for us to figure out is what the variance of this noise is. In the training phase, we estimated **σ**, containing the standard deviation of the noise in each voxel over time. However, we now have a signal that, for each voxel, is an average over multiple time points, and therefore less noisy. So we have to adjust our estimate of the noise variances as follows:

(15)

which, if there equally many time points in the attended and unattended conditions (i.e. ), simplifies to:

(16)

This final piece of the puzzle allows us to estimate **s**’ through (generalized) graphridge regression:

(17)

where . Note that we’re using **W** and **G** here rather than **W**+ and **G**+, because as we saw in equation (11), the baseline activity of our voxels was subtracted out when we computed the difference pattern **b**(D). To select the best value of *λs*, we can use a cross-validation across sets of voxels.

In terms of the probabilistic interpretation, we can define the posterior over **s**’ as follows:

Note that I’m defining it here proportionally, up to a constant of multiplication, since the normalization constant for the likelihood would have to be determined numerically anyway.

# Units & normalization

An important assumption implicit in our generative model is that the bold signal always has the same units. In practice, we know that there are susceptibility differences over time, such that the BOLD signal in one run can have a higher amplitude than in another, without any underlying neural differences. This becomes problematic if we want to generalize from the training to the testing phase, and from pre- to post-learning. We need to know that a signal increase of 1 unit always corresponds to (approximately) the same thing. So we need to normalize the activations with respect to a common baseline. Since we have fixation periods in all runs, that seems like the best thing to use.

First question then is how to do this. Our fixation periods are not so long that we can simply compute the average BOLD in the fixation blocks; that average would actually be a mix between fixation baseline and activation carrying over from visual stimuli in the preceding time points, and crucially, this mix would not be the same between conditions. So the only way we can get a fixation baseline, in both cases, is to fit a model that explicitly models the activation as a baseline + stimulus-evoked activity.

Second question: how do we deal with baselines that are different between runs? Do we fit a model that accounts for this, and then

1. Note that the order of the dimensions is somewhat arbitrary. Somewhere, we have to choose what we want in the rows and what in the columns, and I figured I wanted to have time in the column dimension, which automatically determines the dimensionality of the other matrices. [↑](#footnote-ref-1)
2. Note that we have also cast the convolution with the HRF in terms of a matrix multiplication. This is mostly to keep the notation clear; in the implementation you could compute the result of this convolution by whichever method is most convenient (e.g. Matlab’s conv function) – you don’t have to find the matrix **H** and do the computation as a matrix multiplication. [↑](#footnote-ref-2)
3. This is just a little trick. The multiplication **c1T** simply results in **c** being copied *N* times along the time dimension (since each voxel’s baseline is constant over time). So it’s like using repmat in Matlab. [↑](#footnote-ref-3)
4. Throughout this document I’m using different subscripts to distinguish between different lambda’s, just to make it clear that they refer to different things, stemming from different priors. [↑](#footnote-ref-4)
5. Note that this is different from the cross-validation in the testing procedure, where we split the data up in different sets of voxels rather than time points. This becomes intuitive when you think about how you expect things to generalize. A good pRF estimate should generalize over time: it should be able to predict its voxel’s timecourse in the training set as well as in the validation set. It would not be expected, however, to predict the timecourses of other voxels (obviously). A good stimulus estimate, on the other hand, would be expected to predict the responses at a given time point of voxels in the training set as well as voxels in the validation set (because all voxels responded to the same stimulus), but wouldn’t predict the responses at a different time point (unless the stimulus at that time point was the same). [↑](#footnote-ref-5)