

Dayan and Abbott Notes

Rory Bedford

August 2024

These notes are written for the neuroscience reading group at the LMB, and are therefore aimed at biologists, with a less quantitative background. Some topics within Dayan and Abbott will be mathematically tricky for these readers, so my intention here is to give an overview of many of the topics covered to make things more digestible. Focus is therefore on understanding concepts broadly and how they relate to actual neurobiology, without getting too bogged down in derivations. That said, some familiarity with linear algebra, multivariate calculus and probability theory are still prerequisites for this. If you need a reference for these, I would highly recommend the first half of the textbook *Mathematics for Machine Learning*. The appendix of Dayan and Abbott is also a fantastic resource.

The book is split into three main sections. In the first section, we look at how neurons encode information about the environment. In the second section, we look at biophysical models of neurons such as the Hodgkin-Huxley model, and work our way up to modelling biophysically plausible neural networks. In the final section, we look at learning, including mathematical models of plasticity, some basic reinforcement learning, and some Bayesian inference methods.

Note that my discussion for each chapter does not follow the material in the book in order but jumps around a little. I therefore reference the corresponding section in the book.

1 Chapter 1 - Neural Encoding I

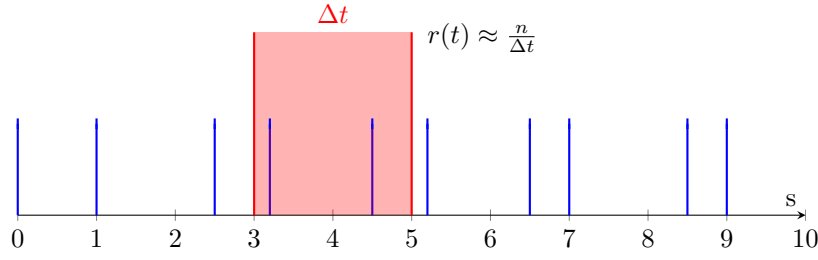
In this chapter, we first look at neuron spike trains and firing rates, which are two different ways of thinking about a neuron's activity. We look at the relationship between these two forms, and how to convert between the two. We also have a first look at neural encoding, with neuron tuning curves and spike-triggered averages.

1.1 Spike trains and firing rates

Neurons fire action potentials at discrete points in time. If we measure a neuron's activity in a single trial, we end up with a list of times t_i for $i = 1, 2, \dots, n$ for n spikes, called a spike train. However, neurons don't work with infinite precision; there is some noise inherent in the timings of these action potentials. We therefore seek an alternate description of a neuron's activity that describes the *rate* at which it is firing, not the exact times of its spikes. The spike-count rate is the neuron's firing rate measured across the entire trial. For a trial of length T :

$$r = \frac{n}{T}$$

The problem with this is that it doesn't account for variation of a neuron's firing rate within a trial, which is an important aspect of how neurons encode information. We therefore seek a firing rate *function*, that varies with time, $r(t)$. This function is an abstraction that is a very useful way to think about neural activity given that spike trains are necessarily stochastic. We can think of it as being the same as the spike-count rate as above, but instead of considering the entire trial, we bin the trial into small windows of length Δt . We can then think of the function $r(t)$ as being the firing rate in the window t to $t + \Delta t$, ie., the number of spikes in this window divided by Δt . This is not very useful if we are only considering one spike train, but if we perform the same recording across many trials, we can average their results to get a good description of a neuron's activity. Furthermore, the more trials we have, the smaller we can make the window Δt , and the more precise of firing rate function becomes.



If we let the window size get infinitely small, then each window either catches a spike, or it doesn't. In this case, the firing rate is zero everywhere, apart from at the exact locations of a spike, at which it is infinite. The function that describes this is the dirac delta function $\delta(t)$. This function has the important property that it integrates to 1, and can therefore be used to 'pick out' values of a continuous function from inside an integral:

$$\int dt' \delta(t - t') f(t') = f(t)$$

We can now represent our spike train as a continuous function $\rho(t)$, rather than just a set of spike times, given by:

$$\rho(t) = \sum_{i=1}^n \delta(t - t_i)$$

For clarity, $\rho(t)$ is the particular function that we measure when recording a neuron's spiking activity. This is generated stochastically from the true underlying firing rate $r(t)$, which doesn't spike but gives a varying firing *rate* over time.

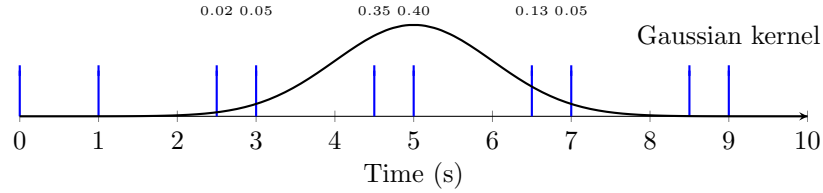
We now need to look at how to convert between these two representations of a neuron's activity - ie, how to estimate $r(t)$ from a set of measured spike trains $\rho(t)$, and additionally, how to sample a spike train from a firing rate function.

1.2 Spike train smoothing

If we could record an infinite number of trials, then our simple window method above would converge to the true firing rate as the window size gets infinitely small. In practise, however, we generally need to estimate the firing rate from a finite number of trials. To do this, we smooth out our set of spike trains by performing a *convolution* with a *window* function $w(\tau)$. A convolution refers to the process of integrating our function of interest with a small sliding window function, sometimes called a kernel, as follows:

$$r(t) \approx \int_{-\infty}^{\infty} d\tau w(\tau) \rho(t - \tau)$$

If you think carefully about this equation, you see that at time t , we centre our window function at t , then compute the values of the window function at all neighbouring spikes, and add their values to give an estimate of the firing rate, as shown below. Also note that in practise, you can average this method over many trials where you have recorded the same neuron to get better results.



$$r(5) \approx 0.02 + 0.05 + 0.35 + 0.40 + 0.13 + 0.05 \\ \approx 1\text{Hz}$$

Since our Gaussian window function varies smoothly, the effect is to smooth out the firing rate function. Smoothing the firing rate captures the fact that a spike train is generated stochastically from the firing rate - the smoother we make it, by widening the window width, the more we get a general rate, and the less the precise spike timings matter. The effects of using different window functions is shown in Figure 1.4 in the book.

1.3 Poisson processes

Having seen how we can estimate the firing rate from a set of recorded spike trains, we also want to consider how spike trains are actually generated from a given firing rate function.

First of all, note that we can easily calculate the mean number of spikes in a trial, as follows:

$$\mathbb{E}[N] = \int_0^T r(t)dt \\ = rT \text{ if } r \text{ constant}$$

However, this tells us nothing about the distribution of spikes around this mean rate. The most basic way to generate spikes is to use a Poisson process. Consider the probability of there being a spike in the window t to $t + \Delta t$. As $\Delta t \rightarrow 0$, the probability of there being 2 or more spikes in this box falls to 0, so we only need to consider the possibility of there being 0 or 1 spikes in this box. This is just a coin flip, or Bernoulli trial, with the probability of there being a spike being given by $P(\text{spike}) = r(t)\Delta t$. Computationally, you could simulate a spike train by splitting your trial into a finite set of small boxes of size Δt , and putting a spike in each box with this probability.

In the case of a homogeneous firing rate - that is, one that is constant for the whole trial, we can say a bit more about the distribution of the number of spikes in a trial. This is done by counting all the different ways we can get n spikes in a finite set of boxes in a trial using combinatorics, and adding their probabilities, then letting our box sizes tend to zero, while using something called Sterling's approximation to give the following nice result:

$$P[N = n] = \frac{(rT)^n}{n!} \exp(-rT)$$

Which is the Poisson distribution with mean rT .

An issue with this simplified model is that we assume the probabilities of a spike being in any bin is independent of the probability of a spike being in any other bin. This is obviously false, and the most major way this is violated is due to a neuron's refractory period - that is, if it spikes, the immediately following bins are extremely unlikely to also spike, regardless of how high the firing rate is. More complex models are able to incorporate a refractory period - in particular, the book discusses methods that sample interspike intervals to generate spike trains. For the homogeneous poisson process, the interspike interval follows an exponential distribution (the derivation in the book is quite straightforward), but this can be modified to a gamma distribution, which can make it almost impossible for a neuron to spike in the refractory period following a different spike.

1.4 Tuning curves

So far, we've only considered a neuron's activity in isolation. We now begin our exploration of how a neuron can encode information about a stimulus. Chapter 1 really only makes a cursory first pass at this, but it gives a good flavour of what's to come.

First of all we consider the concept of a tuning curve. Tuning curves capture the relationship between the values of a stimulus and the mean firing rate of a neuron that encodes this stimulus. They are therefore a useful way of characterising the selectivity of a neuron to sensory information.

Here, we make the simplification that a neuron only encodes information via its mean firing rate - ie, there is no temporal encoding of information, only $\langle r \rangle$ matters. We consider a parameterised stimulus s . For example, the book shows a moving bar in Figure 1.5(A) with its angle of rotation as the parameter of the stimulus. We then simply hold this parameter constant and record the neuron's activity, and calculate its mean firing rate over a trial of given duration. We can vary the value of the parameter(s) across different trials, to build up a picture of how the mean firing rate varies as the stimulus varies. We can then perform any type of curve fitting we want to this dataset to obtain a tuning curve.

This is a useful but slightly limited thing to do. It tells us nothing about how a neuron's activity varies around its mean firing rate. Some of this information could be very useful - for example, think of a neuron in an oscillatory system; information could certainly be encoded in the frequency of its response. A tuning curve would assign the same response to neurons with wildly different oscillating frequencies and would completely miss this encoding. It also can't account for temporal changes in parameters of the stimulus during a trial. Furthermore, many natural stimuli would be almost impossible to effectively parameterise - it therefore only really applies to simple stimuli such as gratings for vision.

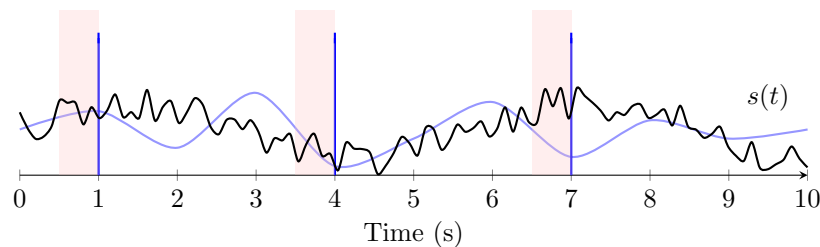
1.5 Spike-triggered average

Another fundamental concept is the spike-triggered average. This is a way of characterising what type of stimulus is most likely to drive a spike. At present we will just define what a spike-triggered average actually is; justification for why this is useful comes in the next chapter when we study receptive fields, which essentially encompass these ideas and tuning curves.

At present, we once again have a parameterised stimulus, which we now allow to vary with time, $s(t)$. We won't worry exactly how it varies - this will be considered in the next chapter. All we need to do to compute the spike-triggered average, is to record a neuron's activity alongside this varying stimulus in a trial. We then pick a window of finite time, and take the mean of the changing stimulus across all the windows preceding all spikes in the recording. We then average this result over any trials. This is described by the following equation:

$$C(\tau) = \left\langle \frac{1}{n} \sum_{i=1}^n s(t_i - \tau) \right\rangle$$

Equation (1.20) also covers the integral forms of this equation in terms of $\rho(t)$ and $r(t)$. It's actually really useful to look at these to check your understanding of our previous discussion on firing rates vs spike trains. Below also shows a visual representation of the spike-triggered average. All we do is take the stimulus in the windows preceding each spike, as shaded, and average them, to obtain $C(\tau)$.



2 Chapter 2 - Neural Encoding II

In this chapter, we study neuron receptive fields and reverse-correlation techniques in detail. That is, we build models that can predict a neuron's activity given an arbitrary time-varying stimulus. These models work well for neurons in the early visual system, which is therefore discussed in detail. In the notes, I try to simplify the explanation of what reverse-correlation techniques are, but do not delve into all the examples of different types of receptive fields given in the notes, as the book describes these far better than I could in the notes.

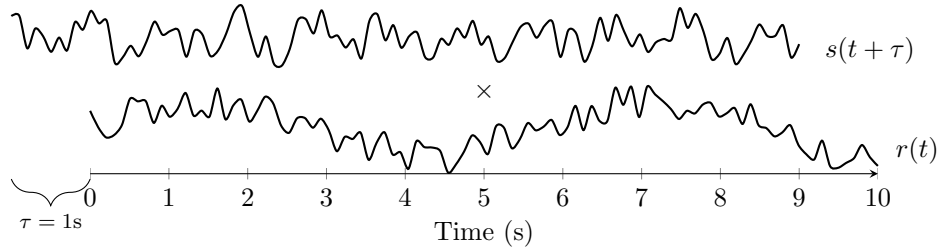
2.1 Correlation functions

Before discussing reverse-correlation techniques, we need to briefly clarify some concepts from chapter 1.

To measure how much two functions correlate, we can multiply them and integrate over all of time. If both functions correlate, then we expect this integral to be positive, since when one function is positive, we expect the other function to also be positive, so they multiply to a positive number, and when one function is negative, we expect the other function to be negative, so they also multiply to give a positive. If this integral is negative, it means that on average the functions have opposing signs, and if the integral is zero, then they are uncorrelated - the value of one function tells us nothing about the value of the other function.

Say we want to compute how much the firing rate of a neuron correlates with a stimulus over the course of a trial. We are interested in the correlation of the neuron's activity with the stimulus at the same point in time, but also the correlation of the firing rate with values of the stimulus preceding the neuron's activity, since the neuron is influenced by the stimulus over a window of time prior to it responding. Therefore, we perform this correlation over successive shifts of the stimulus in time, and record the value as a function of this shift, rather than a single value, like so:

$$Q_{rs}(\tau) = \frac{1}{T} \int_0^T dt r(t) s(t + \tau)$$



We note the relationship here to the spike-triggered average. For the spike-triggered average, we average the stimulus in a window preceding each spike. Here, we average the stimulus preceding the firing rate, weighted by the firing rate. You see that they are essentially equivalent, since the firing rate is what underlies all spike trains. The only differences are that one is normalised by time, and the other by spike count, and also, that the shift τ is defined the other way around.

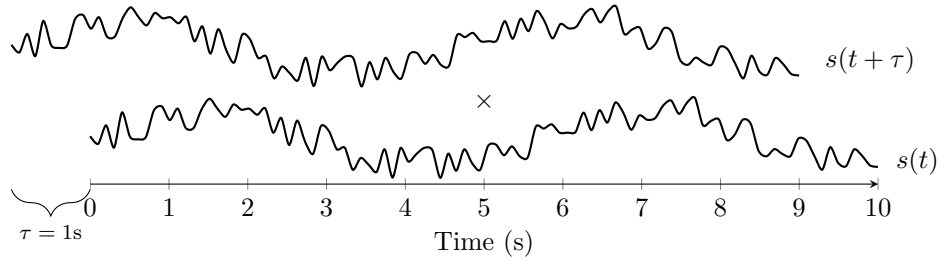
Rewriting the spike-triggered average in integral form (recalling that ρ rewrites a spike train as a continuous function):

$$\begin{aligned} C(\tau) &= \frac{1}{\langle n \rangle} \int_0^T dt \langle \rho(t) \rangle s(t - \tau) \\ &= \frac{1}{\langle n \rangle} \int_0^T dt r(t) s(t - \tau) \\ &= \frac{1}{\langle r \rangle} Q_{rs}(-\tau) \end{aligned}$$

It is also useful to define the stimulus autocorrelation function, which is the correlation of the stimulus with itself:

$$Q_{ss}(\tau) = \int_0^T dt s(t)s(t+\tau)$$

The following graph shows how we shift the function, multiply it by itself, and integrate over time.



Of particular importance is a white noise stimulus. This is a stimulus that is completely uncorrelated with itself - knowing its value at one time point tells you nothing about its value at any other timepoint. Its autocorrelation function is therefore zero for almost all values of τ - except for at zero, where the integral actually blows up to infinity. We represent this with a dirac delta function, which we met when studying spike trains. This is a standard result in signal processing, but understanding it fully requires diving into Fourier analysis, which we don't want to do now, so we just state that the following is the autocorrelation function for a white noise stimulus, where σ_s is the signal's power:

$$Q_{ss}(\tau) = \sigma_s^2 \delta(\tau)$$

2.2 Reverse-correlation methods

We now have the correct toolkit to start studying how to predict a neuron's activity given the values of a stimulus that changes over time. This is actually a regression problem, and we will look at what amounts to linear regression. In regular linear regression, you have a finite set of input vectors and target values $\{(\mathbf{x}_i, y_i) | i = 1 : n\}$. We want to find a weight vector \mathbf{w} and bias b that can be used to estimate the target from an unseen input vector as follows:

$$y \approx \mathbf{w}^T \mathbf{x} + b$$

The solution will only ever be approximate as we assume there to be Gaussian noise added to all measurements. We solve this by minimising the least squares error on our available data:

$$\mathbf{w}, b = \operatorname{argmin}_{\mathbf{w}, b} \sum_{i=1}^n (y_i - \mathbf{w}^T \mathbf{x}_i - b)^2$$

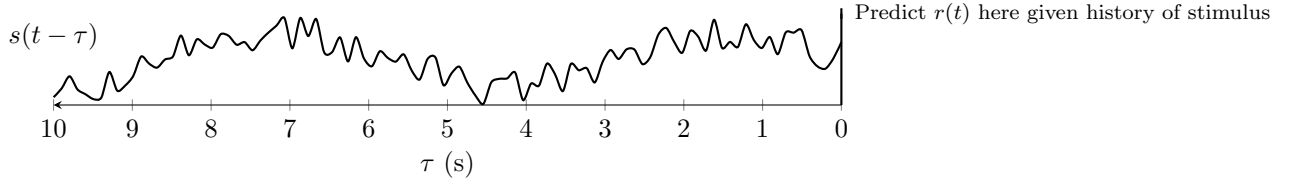
In the special case of whitened input data, which means that the input data has been normalised to have zero mean and covariance proportional to the identity matrix, the solution to this problem is given by:

$$\mathbf{w} = \frac{1}{n\sigma^2} \sum_{i=1}^n y_i \mathbf{x}_i$$

$$b = \frac{1}{n} \sum_{i=1}^n y_i$$

Where σ^2 is the input data variance. (Note this is also tractable in the case of non-whitened data, which is the solution you will typically see in textbooks. Also note that typically 'whitened' also means $\sigma^2 = 1$ but I keep a symmetric variance here for analogy later.)

Now, in the case of neural encoding, we want to calculate the firing rate of a neuron at time t , $r(t)$, given the value of the stimulus in a window leading up to the time t , as follows:



Now, the best way to think about this problem is that the values of $s(t - \tau)$ in the window leading up to the time t at which we want to predict the firing rate form the input vector for our linear regression, and $r(t)$ itself forms the target scalar. It may seem strange that we are referring to a function as a vector, however, this is actually very common. You should think of the vector \mathbf{s} as being labelled by a continuous variable τ , so it has continuously many components, compared to a column vector \mathbf{x} which is discretely labelled. Additionally, t takes the place of the index i in the linear regression problem - where before, we had a discrete number of input vectors and targets, here we use the continuously indexed $r(t)$ as targets with the stimulus values in the window leading up to t as the input vectors. So there are two senses in which we have extended linear regression to a continuous domain.

$$\begin{aligned} \mathbf{x}_i &\rightarrow s(t - \tau) \\ y_i &\rightarrow r(t) \\ i &\rightarrow t \\ \tau &\rightarrow \text{column index of } \mathbf{x} \end{aligned}$$

Many equations from the discrete case can be translated into the continuous case by replacing the appropriate sum over the vector components with an integral over the corresponding function's continuous label. Therefore, a dot product between two vectors becomes an integral of the product of the two functions. We therefore replace the weight matrix \mathbf{w} with a function $D(\tau)$. This is referred to as the filter.

$$\begin{aligned} D(\tau) &\rightarrow \mathbf{w} \\ r_0 &\rightarrow b \end{aligned}$$

The linear equation we want to solve for therefore becomes the integral of the weight function times the stimulus function, integrated over the the stimulus' history:

$$r_{\text{est}}(t) = r_0 + \int_0^\infty d\tau D(\tau) s(t - \tau)$$

And our training dataset becomes the continuous set of all such $(r(t), s(t - \tau))$ pairs present in the recording. Just like before, we minimise the least squares error on the training dataset, which takes the following form:

$$E = \frac{1}{T} \int_0^T dt (r(t) - r_{\text{est}}(t))^2$$

Note also how the normalising constant has changed from the number of datapoints N to the continuous length of the trial T .

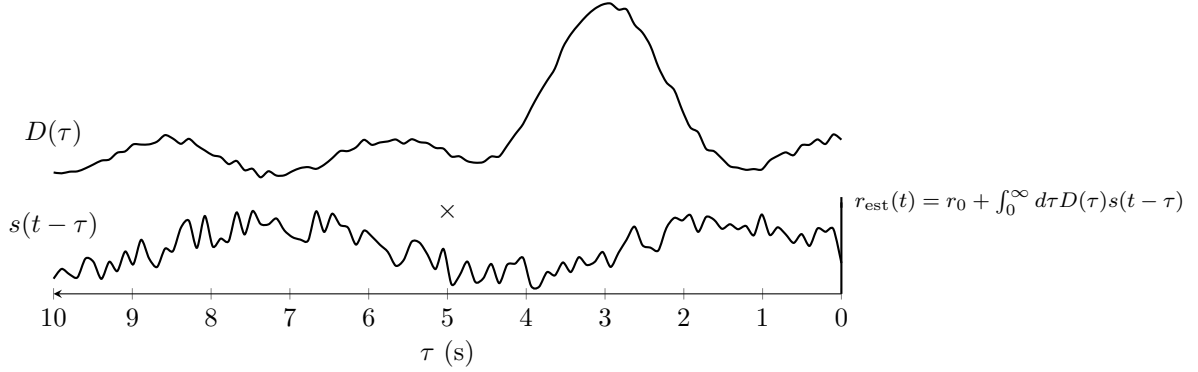
Minimising this equation properly is non-trivial, and requires something called a functional derivative from a subject called the calculus of variations. However, in this case (and definitely don't assume this will always be true!), our solution holds in direct analogy with standard linear regression for the specific case of a white noise stimulus. Previously, we whitened our input data, so that the vectors \mathbf{x} have zero mean and symmetric variance. Here, we use a white-noise stimulus, which is the continuous analog of this, giving the solution:

$$\begin{aligned} D(\tau) &= \frac{1}{T\sigma_s^2} \int_0^T dt r(t) s(t - \tau) \\ r_0 &= \frac{1}{T} \int_0^T dt r(t) \end{aligned}$$

Now, this equation should look familiar. It is in fact, almost exactly the spike-triggered average, with slightly different normalising constants. Substituting these gives us:

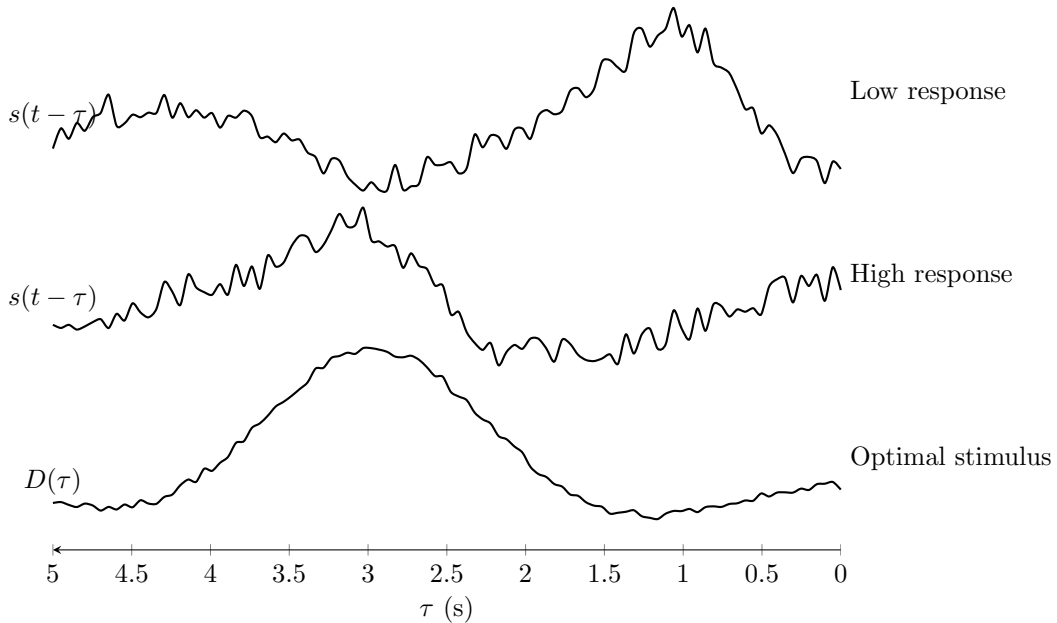
$$D(\tau) = \frac{\langle r \rangle C(\tau)}{\sigma_s^2}$$

This is exactly why the spike-triggered average is a useful thing to calculate. It tells us the optimal linear filter for predicting a neuron's activity from a stimulus.



The case of a non-whitened stimulus is more complex, and requires the use of Fourier analysis to solve properly, which is beyond the scope of these notes. However, it is worth mentioning why a whiten-noise stimulus makes this problem easier to solve. In the non-whitened case, variations of the stimulus in a window leading up to the time at which we want to predict $r(t)$ affect not only $r(t)$, but also each other. Therefore, solving the problem requires completely disentangling the internal correlations in the stimulus, so we can see what directly caused the response, rather than what may have indirectly caused the response through affecting the stimulus value elsewhere.

It's also important to discuss the limitations associated with a linear model. A linear model assumes the response of a neuron is proportional to the amount of overlap its actual stimulus has with its most effective stimulus. We give here a graphical demonstration of what this entails.



We see here that we predict a low response when the stimulus doesn't overlap much with the optimal response. There is in fact no reason apriori to assume this is the case. Later in the chapter, we discuss complex neurons which can have, for example, position-invariant responses to a visual stimulus. A linear model could not capture such a neuron's behaviour: shifting the stimulus in the visual field would cause it to no longer overlap with the optimal stimulus, and hence a linear model

would predict a low response.

We make a quick note here on non-linear models. Do not worry about this too much if you've found this material difficult. The book mentions 3 ways of extending this approach to non-linear models. First of all, it mentions the Volterra/Wiener expansion. This is the functional equivalent of the Taylor series, which allows you to approximate many functions with a polynomial. Including further terms in this expansion is therefore exactly equivalent to do higher-order polynomial regression.

They also discuss using static non-linearities. This is where we compute exactly the same linear filter as before, but pass the output through a pointwise non-linear function, which is optimised to further improve our results. This is different to the full non-linear approach, because the full non-linear approach can model how arbitrary stimuli can affect the response. In the diagram above, both arbitrary stimuli could give high responses, for example, whereas in the static non-linearity approach (assuming a monotonic non-linearity), these would still necessarily get mapped to low and high responses respectively, its just the amount by which the neuron responds is modified in a non-linear way. Finally, the book mentions the use of a non-linearity inside the integral, and in particular, using the response tuning curve for this, so you integrate its static response over time, and develop an optimal linear kernel for that.

Its worth mentioning that many modern approaches to neural encoding will use arbitrarily complex, non-linear models such as deep-learning based approaches. For example, many deeper neurons in the visual system can be modelled well by CNNs.

2.3 Receptive fields

Most of this chapter is concerned with the early visual system. This is because these neurons can be well described by linear models, and are well studied.

So far, we have only studied stimuli that take scalar values. This could include, for example, the total intensity of light in a room. Most real-world stimuli are more complex than this, however, so we need to describe them in more complex ways. A natural extension would be to consider vector-valued stimuli. In this chapter, however, since we are studying the visual system, we actually consider stimuli that vary over a 2D plane, such as a 2D image. Such stimuli can be described by their value not just over time, but over the x,y-axes, like $s(x, y, t)$. Fortunately, all the results from before transfer very easily to this case. For example, the spike-triggered average is defined:

$$C(x, y, \tau) = \frac{1}{\langle n \rangle} \left\langle \sum_{i=1}^n s(x, y, t_i - \tau) \right\rangle$$

And the linear filter is defined:

$$L(t) = \int_0^\infty d\tau \int dx dy D(x, y, \tau) s(x, y, t - \tau)$$

Where the kernel, also called the neuron's space-time receptive field, is given by:

$$D(x, y, \tau) = \frac{\langle r \rangle C(x, y, \tau)}{\sigma_s^2}$$

This can be used just like before to predict a neuron's activity from a stimulus, through either the linear or static non-linear models discussed before. The neuron's receptive field describes the region of sensory space, which in this case consists of 2D images, to which the neuron responds. It is more likely to respond to stimuli similar or overlapping with its receptive field. This might be slightly easier to conceptualise in the case of a separable receptive field, where: $D(x, y, \tau) = D(\tau)D(x, y)$. You can think of $D(x, y)$ as being the image to which the cell is most responsive.

To give a brief descriptive summary of the approach up to this point: first of all, we compute the spike-triggered average from a white noise stimulus, which is the mean stimulus in a window of time preceding each spike. This stimulus varies over both time and space. This gives, up to a constant, the optimal stimulus, ie, the stimulus that is most likely to trigger a spike. To predict a neuron's activity from an arbitrary stimulus, we now simply compute the total amount of overlap

the stimulus has with the optimal stimulus, which estimates the firing rate, with the appropriate constants.

If you've understood the notes up to this point, then the remainder of the chapter should be straightforward, and perhaps more interesting as it dives into the types of receptive fields actually found in neurons in the early visual system. I therefore don't go into detail on this in the notes, as the book does a much better job of this than I could. It does cover interesting topics such as the on-centre and off-centre receptive fields of certain retinal ganglion cells and cells in the LGN, and how neurons in the primary visual cortex can be selective to edges with certain orientations, or even to a moving stimulus and how these receptive fields can be characterised by the linear filters we have discussed. It also briefly covers how you can construct a complex cell which can't be characterised by a linear filter, from a small number of simple cells, to give cells with certain invariances that we previously saw are impossible with a linear filter.