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

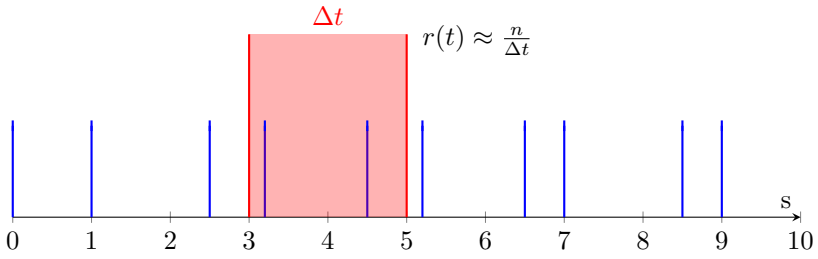
In this chapter, we first look at neuron spike trains and firing rates, which are two different ways of thinking about a neuron's activity. We look at the relationship between these two forms, and how to convert between the two. We also have a first look at neural encoding, with neuron tuning curves 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 $i = 1, 2, ..., n$ 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.

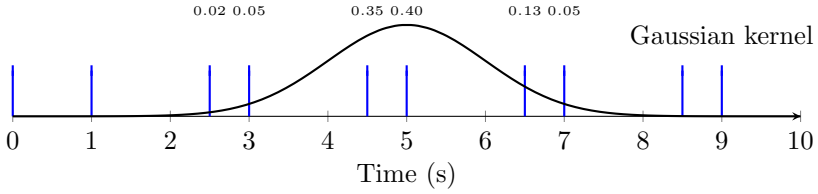
We now need to look at how to convert between these two representations of a neuron's activity - ie, how to estimate $r(t)$ from a set of measured spike trains $\rho(t)$, and additionally, how to sample a spike train from a firing rate function.

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.



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

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:

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

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 $\Delta t \to 0$, 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 being 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 $< r >$ 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.

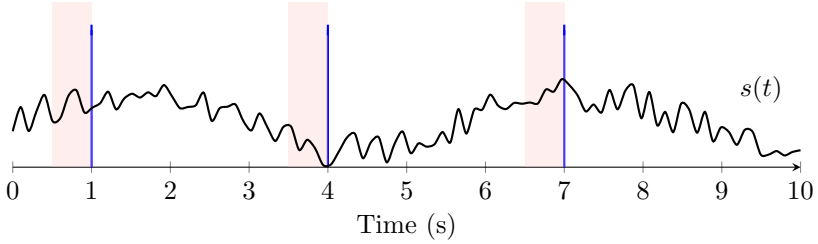
1.5 Spike-triggered average

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 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:

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

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.

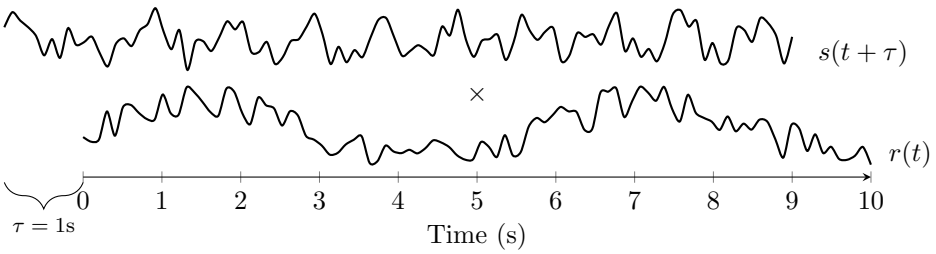
2.1 Correlation functions

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:

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



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.

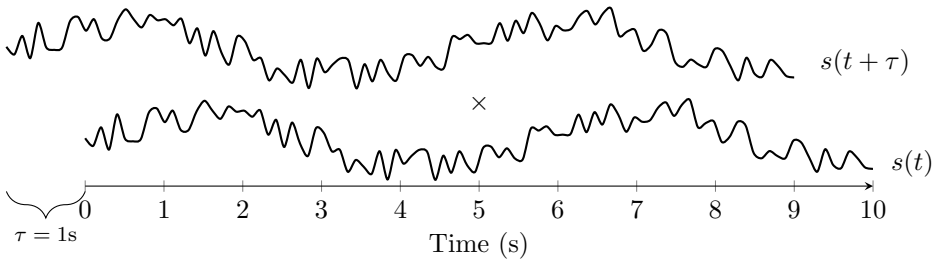
Rewriting the spike-triggered average in integral form (recalling that ρ rewrites a spike train as a continuous function):

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

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 $\{(\boldsymbol{x}_i, y_i)|i = 1 : n\}$. We want to find a weight vector $\boldsymbol{w}$ and bias b that can be used to estimate the target from an unseen input vector as follows:

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

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:

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

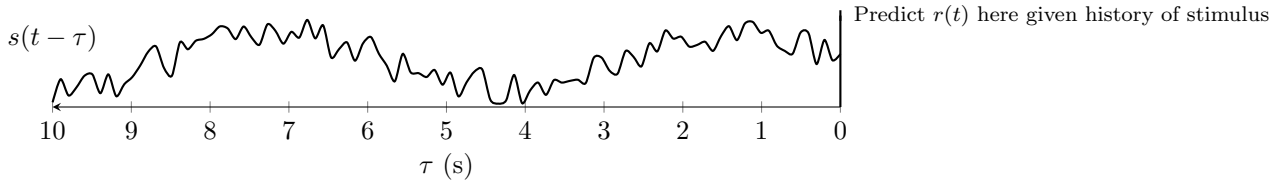
In the special case of whitened input data, which means that the input data has been normalised to have zero mean and covariance proportional to the identity matrix, the solution to this problem is given by:

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

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

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 $\boldsymbol{s}$ as being labelled by a continuous variable τ, so it has continuously many components, compared to a column vector $\boldsymbol{x}$ 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.

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

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 $\boldsymbol{w}$ with a function $D(\tau)$. This is referred to as the filter.

$$\begin{aligned}D(\tau) &\to \boldsymbol{w}\\ r_0 &\to b\end{aligned}$$

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.

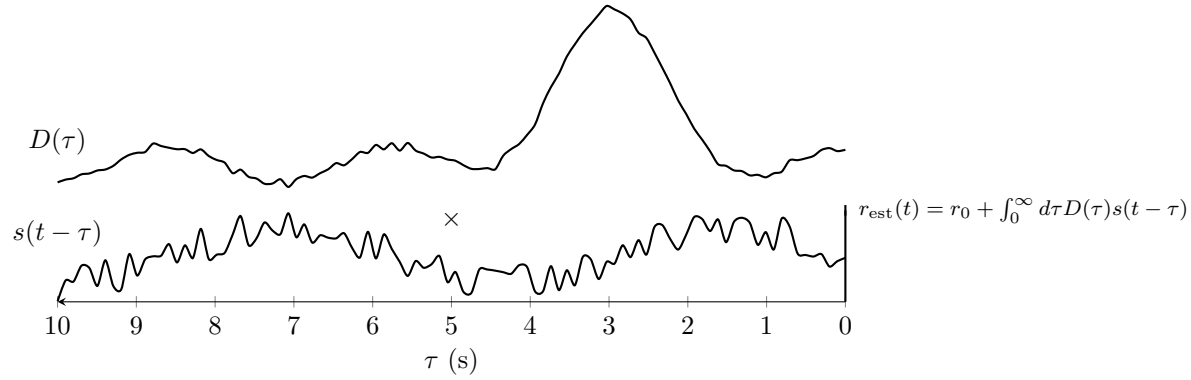
Minimising this equation properly is non-trivial, and requires something called a functional derivative from a subject called the calculus of variations. However, in this case (and definitely don't assume this will always be true!), our solution holds in direct analogy with standard linear regression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

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

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:

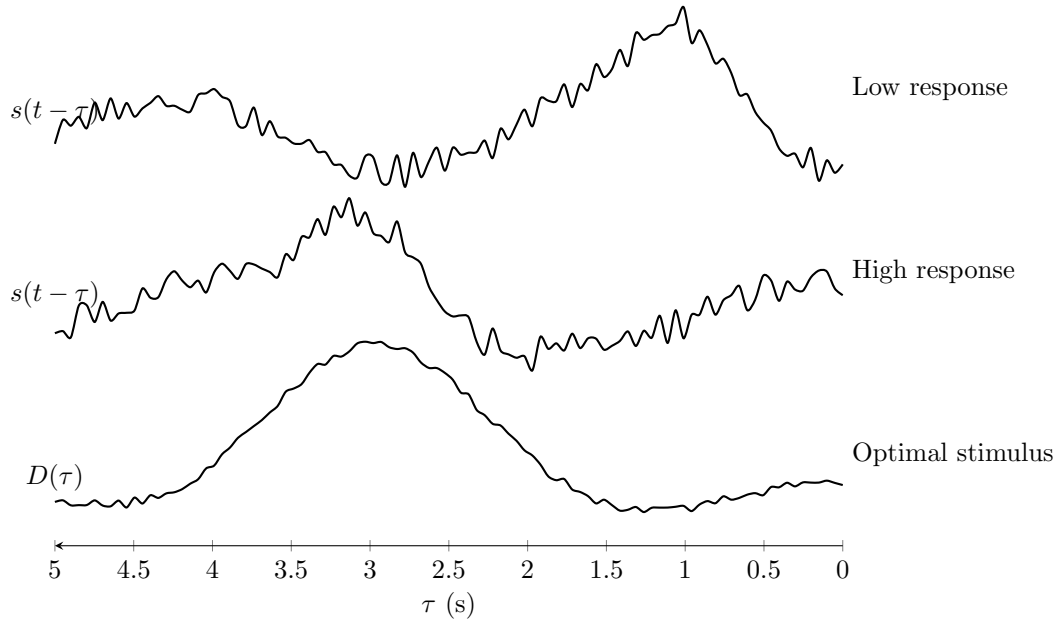
$$D(\tau) = \frac{\langle r \rangle C(\tau)}{\sigma_s^2}$$

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.

3 Chapter 3 - Neural Decoding

Neural decoding is the reverse of neural encoding: its goal is to construct an estimate of the stimuli, given neural activity. This chapter begins by looking at rate-based models, where we predict static stimuli from single-neuron or population spike-count rates. In particular, it begins with two specific models: a single-neuron threshold test, and a vector-based model. These are highly specific to the cells and stimuli they model, so are useful to cover, but not as informative as the development of a generalised toolkit for neural decoding, which is what we will therefore focus on. Rate-based decoding can in general be seen as a classification or regression task, so much of the machine learning literature is directly relevant. We will focus on a particularly powerful approach, that of probabilistic, Bayesian models. This framework gives a theoretically satisfying and unified approach to the problem of rate-based decoding. We will then also discuss Fisher information, which can help develop some intution about how a neuron encodes information about a stimulus. Finally, we will also cover spike-train decoding methods.

3.1 Bayesian rate-based decoding

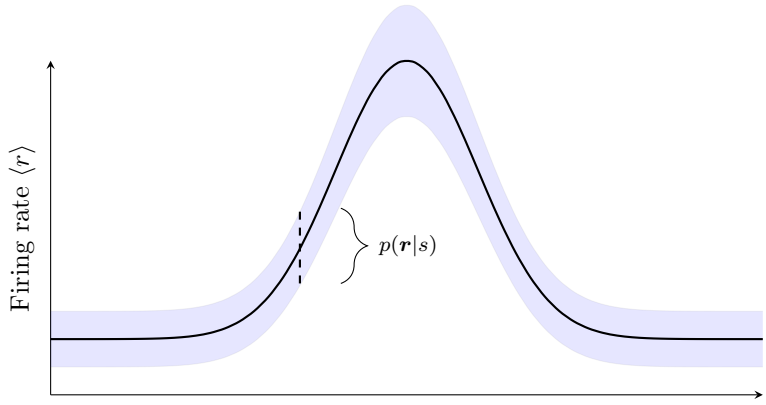
Neural decoding is essentially an inverse problem: in the context of sensory stimuli, we view the stimulus as a cause, with neural activity as a response dependent on this cause. This means that we can fairly directly measure this relationship in the forward direction. As we saw in chapter 1, we can freely change a stimulus to any value we want, measure the spike-count rate of a neuron over several trials, and then directly characterise this relationship with a tuning curve. The inverse relationship is not directly accessible to us in the same way: we cannot fix a neuron's response to a set value and record all the stimuli that caused this response. We therefore need to use inverse techniques to disentangle this relationship.

In particular, we take a probabilistic approach, where we accept from the get-go that there is noise present in both the neural response and our measurements, and then construct optimal estimates of the stimulus from these noisy measurements. The toolkit used to do this is known as Bayesian inference, and provides a unified theoretical framework for the decoding problem, and is also an optimal decoding strategy in the presence of noise.

In the Bayesian approach, we assume a generative model of the neural response given a stimulus, $p(\boldsymbol{r}|s)$. This is a conditional probability distribution, which is a distribution over the possible firing rates of all the neurons being measured, given some value s of the stimulus.

Previously we studied tuning curves, which give the mean firing rate of a neuron given a stimulus. This amounted to just fitting some parameterised curve to the measured r, s pairs, such that $f_\theta(s) \approx \langle r \rangle$. Tuning curves can take many different functional forms, as we have seen. Now all we are doing is extending this idea to the full probability distribution of the neuron's firing rate given a stimulus. Often, we will use the Poisson distribution for this, which, as we saw in chapter 1, gives a good approximation of the distribution of the spike-count rate over the course of a trial, and has the benefit of not needing any additional parameters to fit beyond the tuning curve, as it is characterised entirely by its mean. Other distributions are possible though; for example, you may

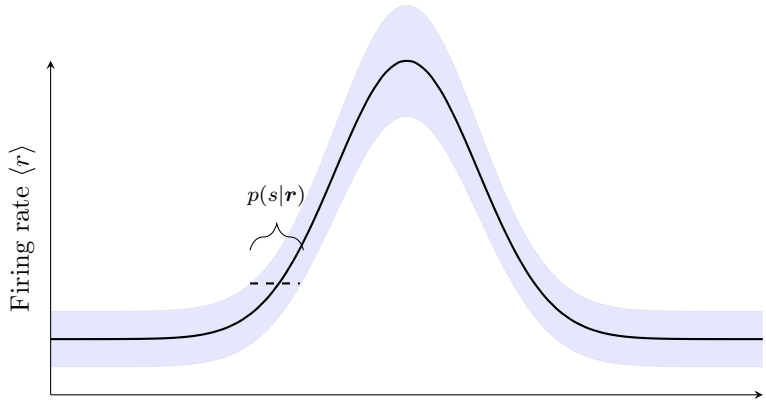
come across tuning curve plus isotropic (uniform) Gaussian noise models, or models where the noise is proportional to the mean firing rate.



Recall, for example, the form of the Poisson distribution, assuming independence between different neurons, and where each neuron a is described by a different tuning curve $f_a(s)$:

$$p(\boldsymbol{r}|s) = \prod_a \frac{e^{-f_a(s)T}(f_a(s)T)^{r_a T}}{(r_a T)!}$$

To be very clear here, the term $f_a(s)T$ is the mean spike-count of neuron a given our tuning curve, and $r_a T$ is the spike-count we actually measure.



So our procedure at this point is to fit the parameters of a tuning curve for each neuron to the measured data, and use this to describe a generative model that gives a distribution of firing rates for each neuron given a stimulus. Now the decoding task is to estimate the stimulus *given* the firing rates. This is done using Bayes' theorem, which states that the posterior distribution of the stimulus given the firing rates is proportional to the likelihood of the firing rates given the stimulus, times the prior distribution of the stimulus:

$$p(s|\boldsymbol{r}) = \frac{p(\boldsymbol{r}|s)p(s)}{p(\boldsymbol{r})}$$

$$\text{Posterior} = \frac{\text{Likelihood} \times \text{Prior}}{\text{Evidence}}$$

The prior term, $p(s)$, represents our 'best guess' distribution over the stimulus values before we have any knowledge of the neural response. For example, if we know the stimulus takes value a two-thirds of the time, and b one-third of the time, then we could set $p(s = a) = 2/3$ and $p(s = b) = 1/3$. If no such information is available, then we typically set our prior to a uniform distribution, or some other distribution that is relatively uninformative - typically, for relatively sensible priors, the results won't be affected too much.

The evidence term is calculated by marginalising over the stimulus, and acts as a normalising constant. In full:

$$p(\boldsymbol{r}) = \int ds p(\boldsymbol{r}|s)p(s)$$

As we can see, dividing by this term just ensures that we have a well-defined probability distribution, as integrals over a probability distribution must be equal to 1.

The full Bayesian approach would be to just leave things here: given a measured neural response, we return the probability distribution over possible stimulus values given by Bayes' theorem. However, there are cases where we require a point estimate of the stimulus, ie, a specific value of s that we determine to be the 'best' estimate of s given $\boldsymbol{r}$. One method of doing this is to take the mode of the posterior distribution - that is, the value of s that maximises the posterior distribution. This is known as MAP estimation (maximum a posteriori). It has the benefit of us not needing to calculate the normalising constant, which is typically analytically intractable, and can be computationally expensive to approximate. Note that we can plug our posterior into any monotonic (always increasing) function, and the MAP estimate will be identical. Typically, we therefore maximise the log posterior, as the logarithm function has nice algebraic properties:

$$s_{MAP} = \text{argmax}_s \log p(\boldsymbol{r}|s) + \log p(s)$$

And we see that we can safely ignore the $-\log p(\boldsymbol{r})$ term, as it does not depend on s.

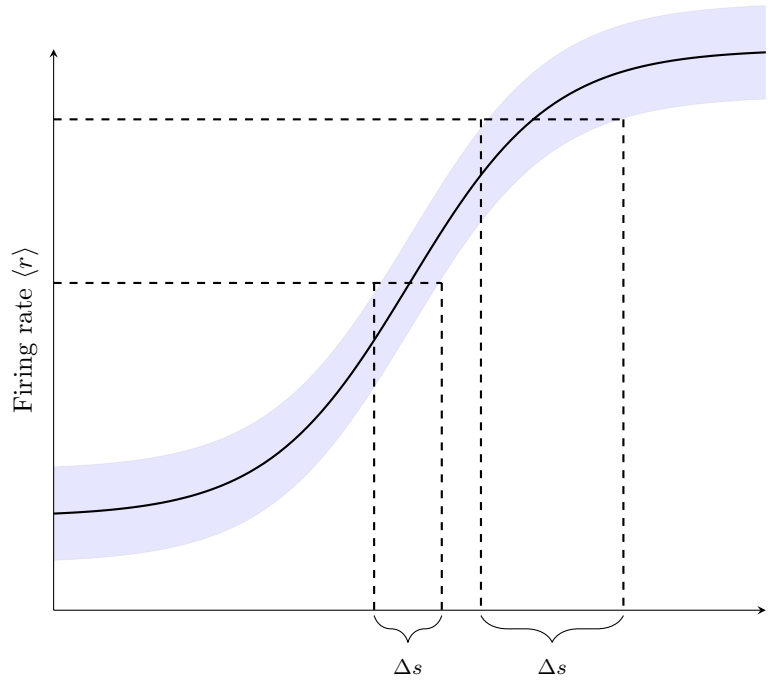
As with many optimisation problems, the solution for many models is determined by differentiating the log posterior with respect to the stimulus, and setting the result to zero. This can give an exact analytical solution for s_{MAP}, into which we can plug in any firing rate vector $\boldsymbol{r}$ and get out our best estimate of the stimulus that caused it. The book covers some examples of solutions where this is the case.

The maximum likelihood (ML) approach is to give instead the stimulus that maximises $\log p(\boldsymbol{r}|s)$. This is identical to the MAP estimate, but ignores the prior, and is therefore equivalent to using a uniform prior that is independent of s.

3.2 Fisher information

Fisher information is a measure of how much information a neuron, or population of neurons, can carry about a stimulus. It tells us the amount of accuracy with which we can hope to decode a stimulus value. It is therefore useful in the theoretical derivation of error bounds for decoding algorithms, which are beyond the scope of these notes. However, a brief discussion of Fisher information is helpful here as it aids our understanding of how neurons encode information. Note that a somewhat more useful measure is given by Shannon's mutual information, which we will discuss in the next chapter.

Consider, once more, the tuning curve of a neuron. Imagine that we want to decode a stimulus value in the presence of noise. The diagram below shows the deviation of our decoding at a region where the tuning curve is steep, and a region where the tuning curve is shallow. The key takeaway from this section is that neurons are maximally informative about a stimulus at regions where their tuning curve is steepest. The diagram makes this obvious, but it may be somewhat surprising at first, as we would naively assume neurons are maximally informative where they are responding the most.



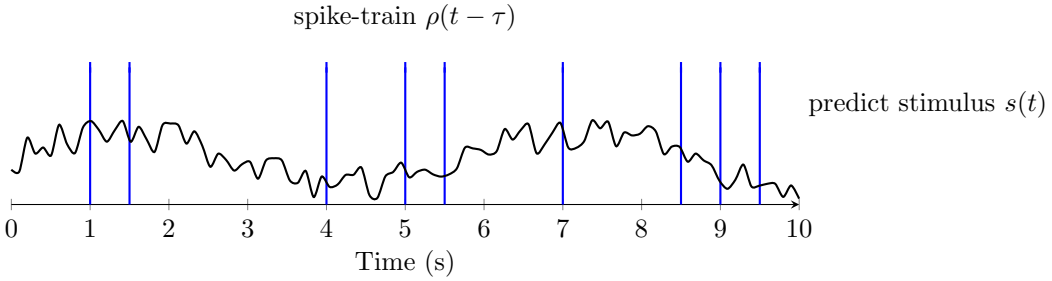
The Fisher information captures this through the term $\left(\frac{\partial \ln p(\boldsymbol{r}|s)}{\partial s}\right)^2$, where the square ensures that it doesn't matter if the slope is pointing up or down, and the logarithm is used for its nice algebraic properties. This is not defined for the mean of the tuning curve as shown in the diagram but for the entire conditional probability distribution, so we then just take the mean of this term over the entire set of possible responses:

$$I_F(s) = \int d\boldsymbol{r} p(\boldsymbol{r}|s) \left(\frac{\partial \ln p(\boldsymbol{r}|s)}{\partial s}\right)^2$$

Plugging in a value of s into the Fisher information gives us a measure of how much information the neural response can have about the stimulus at that point.

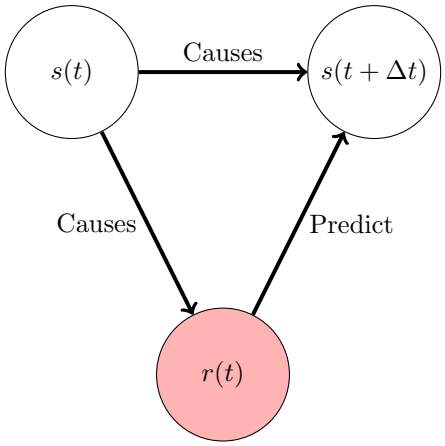
3.3 Spike-train decoding

So far, we have only considered rate-based decoding methods. If we want to decode a time-varying stimulus, then we need to use spike-trains. A basic statement of this problem is that we are given as input a series of spike-times $(t_1, ..., t_n)$ up to a time t, and we want to predict the stimulus value at this time, $s(t)$. Recall that we often describe a spike-train using a sum of dirac-delta functions to give a continuous function $\rho(\tau)$. In the diagram below, note that τ is the time parameter we use to describe the history of the trial so far, which we can use to predict the stimulus at the current time point t.



This problem is in fact almost exactly the reverse of the time-varying neural encoding problem we met in chapter 2. However, there are a couple of crucial differences that warrant discussion.

As we've already discussed, in a laboratory setting, we can treat the stimulus as an independent variable, which causes the dependent variable that is the neural response. This means we can give the stimulus any structure we want, such as making it a white noise stimulus, which is how we solved the encoding problem. When decoding, however, we can't directly control the neural response, so we have to work with whatever response we are given. Furthermore, in the time-varying case, the direction of causation makes our problem statement quite unusual: the stimulus causes future values of the response, yet we are trying to predict the stimulus from the response's history. This is, of course, only possible if the history of the *stimulus* provides information about the future stimulus, as the neural response can in reality only tell us about the stimulus that caused it, not the future stimulus.



In the diagram above, we can only observe the response, but can use this to predict the stimulus at a later time, as they are still correlated via the stimulus history. This does mean, however, that spike-train decoding is impossible if the stimulus is white-noise: we require internal structure within the stimulus.

A classic application of spike-train decoding methods would be a brain-computer interface, where we are trying to gain information about a stimulus on the fly from neural activity. For this reason, we talk about spike-train decoding, rather than firing rate decoding: we want to be able to form a prediction for a single trial, as we won't necessarily be able to record many trials to smooth over to give the firing rate function. Additionally, note that we introduce a time lag τ_0 into our decoding, as this increases accuracy, as it allows *some* spikes to occur after the stimulus we are decoding, and therefore be casued by them. However, too large a time lag would make many BCI applications impossible.

Proceeding exactly like the encoding case, we want to form a linear estimate of the stimulus at time $t - \tau_0$, given the entire spike-train up to the time t. This is done by integrating over the whole trial with a kernel $K(\tau)$. Note in the equation shown, the integral is over all time, not from time 0 to infinity as you might expect. This means technically we are considering spikes after the time t (recall τ runs backwards from time t). This can be avoided by making the kernel acausal, which means we clamp its value to 0 for negative values of τ.

$$s_{est}(t - \tau_0) = \int_{-\infty}^{\infty} d\tau(\rho(t - \tau) - \langle r \rangle)K(\tau)$$

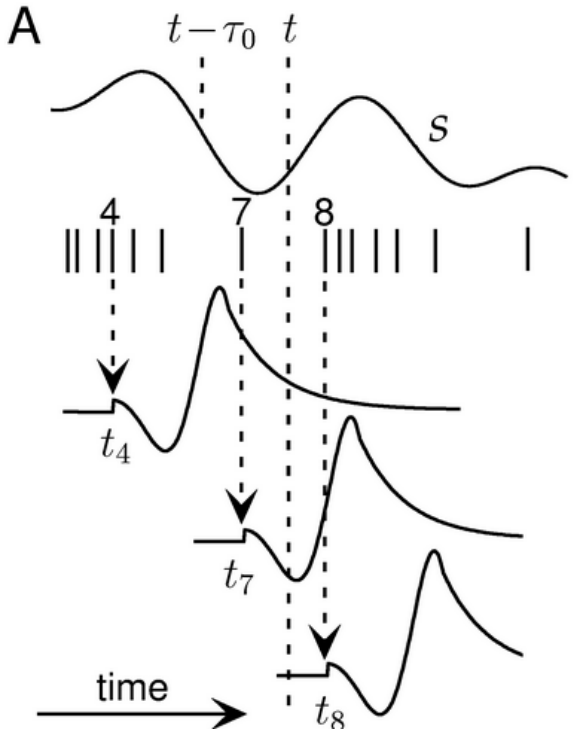
Our job is to figure out the kernel that gives the *best* estimate of the stimulus given the spike-train. Just like with neural encoding, we consider a training set of trials where we measure both the spike-trains and the stimulus, and minimise the following mean squared error over all these trials:

$$\text{MSE} = \frac{1}{T}\int_0^T dt \left\langle (s(t - \tau_0) - s_{est}(t - \tau_0))^2 \right\rangle$$

We can't minimise this like before, because the neural response is not white noise (it has a non-delta autocorrelation), so the full solution requires a Fourier decomposition which is beyond the scope of these notes. However, in the case of a low firing rate, where effects such as the refractory period don't matter much, our spike-trains effectively have zero auto-correlation and the solution is

very interpretable, so we briefly mention what the solution amounts to.

Recall the spike-triggered average $C(\tau)$ discussed in chapter 1. Our optimal kernel amounts to the spike-triggered average, defined so that we consider stimuli after spikes rather than before, and shifted an amount τ_0 to the left, to account for the time lag. Our decoding procedure then just sums up the spike-triggered average over all spikes, as shown in the figure below.



This is in fact an extremely intuitive result. If we only measured one spike, our best possible guess for the stimulus' future values would be the spike-triggered average following this spike. In the presence of many spikes, we do the same thing, summing over the best possible guess provided by each spike individually.

4 Chapter 4 - Information Theory

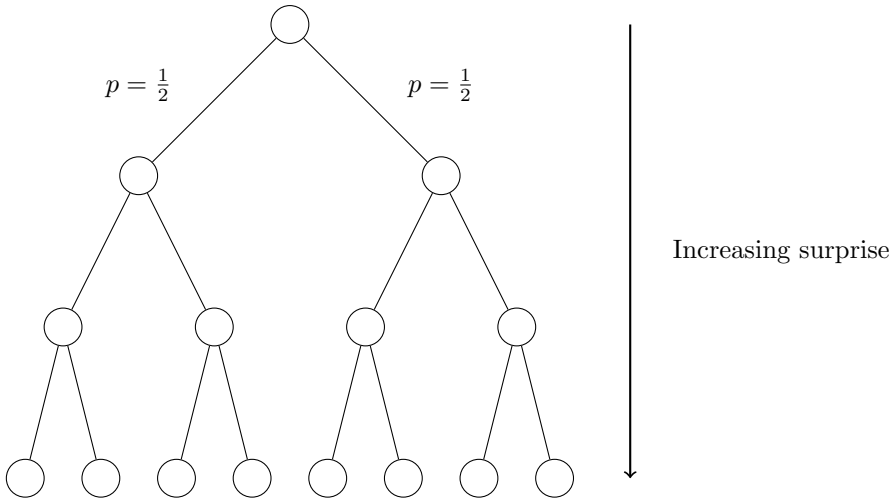
Previous chapters have been interested in *how* neural encoding occurs. This chapter is concerned instead with *what* type of encoding should take place. In particular, we focus on the efficiency of encodings in the very early visual system - that is, how can encodings be designed so that we express as much information as possible about natural scenes. The toolkit we develop in order to answer this question comes from Claude Shannon's information theory, which was originally conceived to describe electronic communication down noisy channels such as telephone wires, but is equally applicable to neural communication channels. So we begin with an overview of information theory, then give a more qualitative discussion of how neurons can maximise their coding efficiency than that presented in the book.

4.1 Information and entropy

A brief overview of probability distributions, information, and entropy by themselves is useful here before we dive into relating these to neural responses and stimuli.

The first concept we will discuss is surprise. Imagine you toss a single unbiased coin, and the result comes up heads. We can characterise the amount of information this result gives you using a quantity called surprise. This is measured in bits - and the information gain from observing heads for a coin toss is 1 bit. So a bit is by definition how much uncertainty is resolved by knowing the outcome of an equal probability binary variable. (It's worth noting, however, that this term is slightly contentious - we often use 'bit' to refer to a binary variable irrespective of whether its probability

is 50/50, such as a bit in your computer). We can extend this idea to arbitrary probabilities as follows. Imagine that you have repeated fair coin tosses: the information gained from each coin toss is additive. The diagram below illustrates this for 3 repeated coin tosses: the total information gain is 3 bits, while the probability of each final outcome is $\frac{1}{8}$.



$p = \frac{1}{8}$ or surprise = 3 bits

A moments thought tells you that the surprise is related to the probability of an outcome via the following formula:

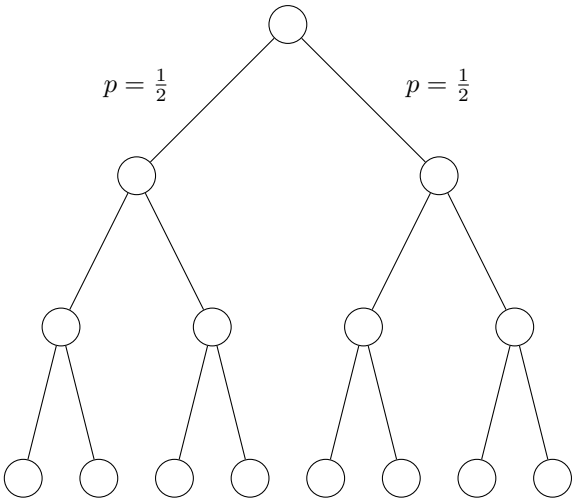
$$\text{surprise} = -\log_2(p)$$

We extend this naturally to probabilities that are not powers of 2, which will lead to non-integer values for the surprise. The surprise of an unlikely event is high, while the surprise of an unlikely event is low. This makes sense - if you observe an event that was certain, ie, has probability one, you have gained zero information, and the surprise is zero, while for very unlikely events, the surprise can grow incredibly high. Surprise as a quantity has the important property that it is additive over non-independent events. This means the information gain from independent coin tosses add up. This can be shown easily, as independent probabilities multiply, and the logarithm of a product is the sum of the logarithms.

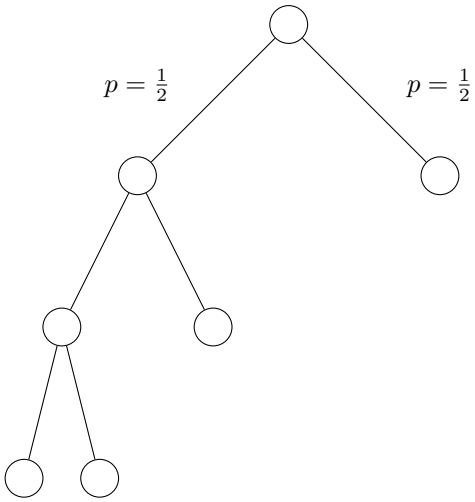
While surprise is a defined for a the measurement of a single outcome, entropy is defined across an entire probability distribution. Entropy simply tells you, on average, how surprising is a measurement drawn from that probability distribution. We usually think of high entropy distributions as being *disordered*, while low entropy distributions are *ordered*. This aligns with our more formal definition: for a very ordered distribution, the outcomes of draws are typically not very surprising, while disordered distributions have more surprising draws on average. Entropy is also measured in bits. Formally, the entropy of a probability distribution is given by:

$$H = -\sum_i p_i \log_2(p_i)$$

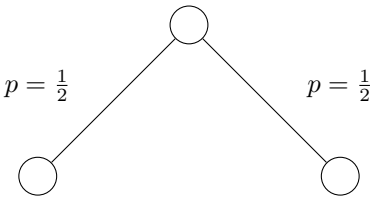
If we go back to our binary tree example, we can graphically represent some different probability distributions with different entropies. We see that the entropy means, on average, how far down the tree do you traverse. For general probability distributions this idea is extended, using the same formula, and really has the same meaning but lacks the graphical interpretability of a binary tree.



$H = 8 * \frac{1}{8} * 3 = 3$ bits
High entropy



$H = 0.5 * 1 + 0.25 * 2 + 2 * 0.125 * 3 = 1.75$ bits
Medium entropy



$H = 2 * 0.5 * 1 = 1$ bits
Low entropy

For continuous probability distributions, we can also define the differential entropy, as:

$$H = -\int dx p(x) \log_2(p(x))$$

There are complications with continuous distributions discussed in the book in terms of having to define a measurement accuracy. In fact, the above definition doesn't really make sense (look at the units - $p(x)$ has units $[x]^{-1}$, and so we're taking a logarithm of a quantity with units, which doesn't make sense, and also it means a this quantity is not invariant under transformations of x). We won't really worry about the details of this, but it's worth being aware of.

Before discussing further topics within information theory such as mutual information, we will quickly review multivariate probability distributions. A probability distribution can be defined over more than one variable. $P(x, y)$ is the joint probability that the first variable takes value x, and the second variable takes value y. Such probability distributions are normalised, such that:

$$\sum_{x,y} P(x, y) = 1$$

We define the marginal probability distribution over x as the distribution that we get if we don't care about measuring the y values - so we sum over all possible y values we could measure:

$$P(x) = \sum_{y} P(x, y)$$

Additionally, we define the conditional probability distribution as the distribution over x, *given* that we know the value of y:

$$P(x|y) = \frac{P(x,y)}{P(y)}$$

The denominator ensures that this is a correctly normalised probability distribution in the variable x, so that $\sum_x P(x|y) = 1$.

In general, x and y are dependent. This means knowing the value of one of them tells us something about the other. Independence is when they tell us nothing about the other, and so:

$$P(x,y) = P(x)P(y)$$

Or equivalently:

$$P(x|y) = P(x)$$

We can now develop our notion of entropy further for multivariate distributions. In general, we want to quantify how much does knowing variable y tell us about variable x. Entropy comes in handy here. It could be that, without knowing y, the distribution over x is very disordered; it has high average surprise. However, knowing y could make the distribution over x very ordered. This would mean that y is very informative about x, and that these two variables have high mutual information. We will formally define these notions now.

First of all, we define the full entropy for the joint distribution, which tells us the average information we gain by measuring both variables together:

$$H(X,Y) = -\sum_{x,y} P(x,y) \log_2 P(x,y)$$

We can also define the entropy of a single variable, ignoring the value of the other variable, via its marginal distribution, which tells us the information we gain if we measure this variable, ignoring the other variable:

$$H(X) = -\sum_{x} P(x) \log_2 P(x)$$

We can also consider the the entropy of the conditional distribution. This tells us the average information we gain by measuring the variable x, given that we know the value of y. Note this would in general be a function of the value of y we measure. We therefore take the expectation over y, which gives what is known as the conditional entropy:

$$\begin{aligned} H(X|Y) &= -\sum_{y} P(y) \sum_{x} P(x|y) \log_2 P(x|y) \\ &= -\sum_{x,y} P(x,y) \log_2 P(x|y) \end{aligned}$$

Note that all the various ways of relating full, conditional, and marginal probabilities also apply to these entropies, however, the logarithm turns multiplication/division into addition/subtraction. For example:

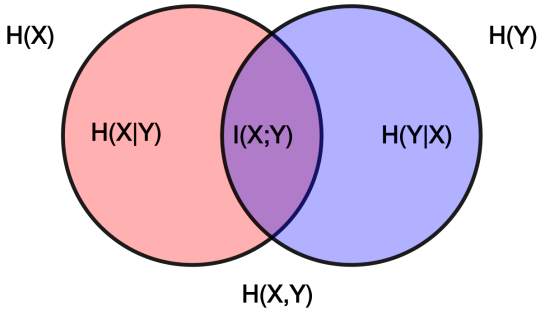
$$H(X,Y) = H(Y) + H(X|Y)$$

A particularly important quantity is the mutual information between two variables. This is defined as the reduction in uncertainty about one variable that is obtained by knowing the value of the other. Our initial uncertainty in x, without knowing y, is just given by the unconditional entropy on x $H(X)$, while the uncertainty if we do know y is given by $H(X|Y)$. The conditional entropy $H(X|Y)$ essentially defines a limit on how much certainty we can have about X by knowing Y - even in the presence of complete knowledge of y there is still inherent noise in the variations of x. It is therefore sometimes defined as the noise entropy. The mutual information is therefore:

$$I(X;Y) = H(X) - H(X|Y)$$

Note this is symmetric in X and Y (try to prove this!). It is also non-negative, as the conditional entropy is always less than the full entropy, and equals zero only when x and y are independent. You can interpret the mutual information as a measure of the mutual dependence between the two variables, or how much knowing one tells us about the other.

The following diagram captures all the useful relationships (areas add or subtract):



4.2 Neural information

We are now ready to apply our knowledge of information theory to study neural encodings. Some of the mathematics in this chapter gets particularly hard, however, the key concepts are actually quite interpretable, so even more so than in previous chapters our discussion will be quite qualitative.

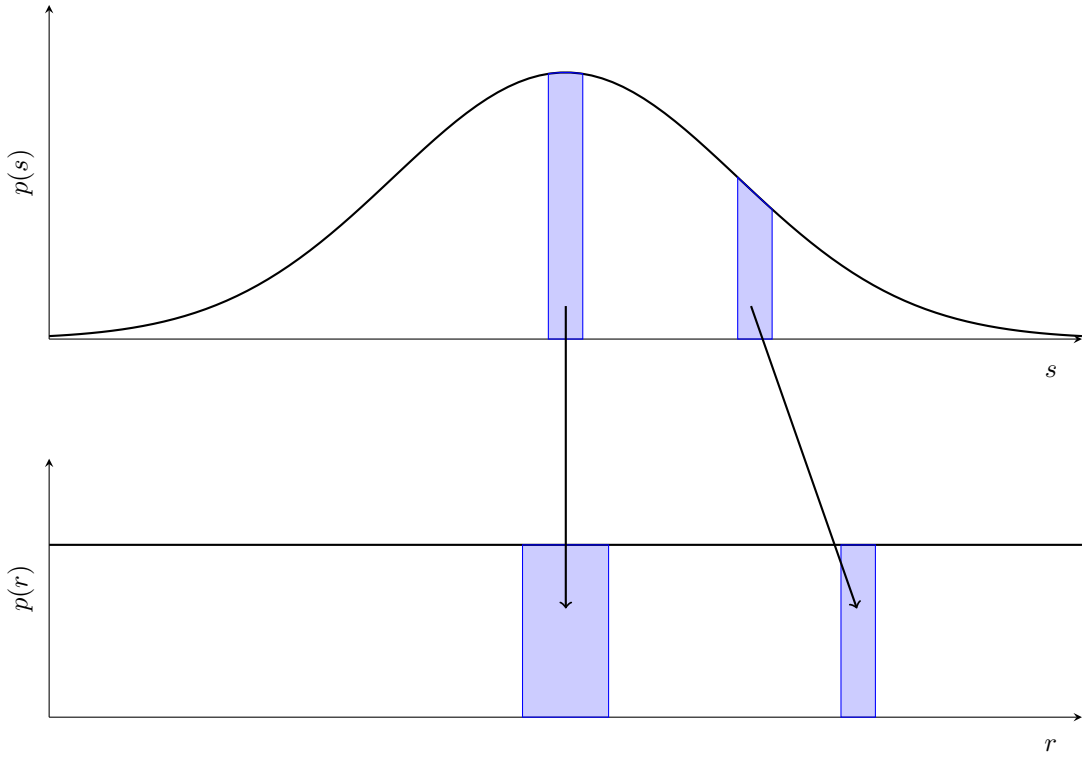
We will consider neurons that use a rate-based encoding to describe their stimulus, meaning they are described by a continuous variable (see the discussion in the book on how our measurement precision of this variable affects the entropy). This neuron encodes a stimulus via its tuning curve (with noise), which gives a conditional probability distribution of the form $p[r|s]$, where $E[r|s] = f(s)$, the tuning curve. We seek to answer the question: what type of tuning curve should the neuron have? In general, neurons of course encode in a way that can solve a variety of problems. A good framework with which to view neural encodings early in the processing chain, such as retinal ganglion cells and cells in the LGN for vision, is to consider them as forming a noisy channel down which we want to send as much information as possible to the primary visual cortex where more complex processing can occur. We therefore take an information-theoretic perspective, in which we want these neurons to express as much information as possible about visual stimuli in their responses. This perspective is quite successful at describing the operation of the early visual system.

We therefore seek an encoding that maximises the mutual information between the response and the stimulus. First of all we will consider the marginal distribution over the response, $p(r)$. Regardless of the stimulus, we can already describe what this should look like. We want this distribution to be as disordered as possible - if every measurement from it is as surprising as possible, then the information we gain from each measurement is as high as possible. This means it should be an entropy maximising distribution. Compare this with the opposite extreme: if our distribution always gives us the same response r, so it has zero entropy, then we gain no inofrmation at all by measuring r! So we must maximise the entropy of $p(r)$ subject to some constraints (without constraints we would try to use all real numbers up to infinity to maximise the entropy). Below we list some common constraints and their corresponding entropy maximising distributions.

- Maximum firing rate r_{max}: The distribution is a uniform distribution between 0 and r_{max}.
- Fixed mean firing rate $\langle r \rangle$: The distribution is the exponential distribution $p(r) = \frac{1}{\langle r \rangle} e^{-r/\langle r \rangle}$.
- Fixed mean and variance: The distribution is the Gaussian distribution $p(r) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(r-\mu)^2/2\sigma^2}$.

The marginal distribution over stimuli is given to us by nature: we have no control over it. The idea of an information maximising encoding is to find an appropriate conditional distribution $p(r|s)$ which maps this given stimulus distribution to the entropy maximising distribution (recall $p(r|s)$ is

essentially the tuning curve plus noise). The mapping that acheives this has an interesting property: it is area preserving. This means that we encode stimuli with a precision proportional to their likely occurrence: very likely stimuli are encoded with high precision, while unlikely stimuli are encoded with low precision. This is a very intuitive result, and can be seen in the diagram below (for the maximum firing rate constraint).



So our distribution satisfies the relationship $p(r)\Delta r = p(s)\Delta s$, or equivalently, the tuning curve satisfies the relationship $\frac{df}{ds} = \frac{p(f(s))}{p(s)}$. The main takeaway to remember here is we want an encoding that makes our stimulus into essentially a white noise response.

For populations of neurons, it's fairly obvious that different neurons must encode different things to maximise the amount of information they represent. In the information maximisation paradigm, we take this to the extreme, and assert that the responses of any two neurons must in fact be completely independent of each other, so there is no redundancy in their encoding. This means the full distribution over all neurons fully factorises: $p(\boldsymbol{r}) = \prod_a p(r_a)$. This makes sense: if any neurons are correlated in their responses, then measuring one tells us information about the others, which reduces the amount of information we can gain from actually measuring the others. The information maximisation approach is of course not true for deeper processing stages: we often end up with highly redundant encodings for a multitude of reasons, but it does provide a good description of the early visual system.

So far we have only considered the spike-count rate of neurons and populations of neurons, however, our discussion applies as well to firing rate based encodings. For the early visual system, we know that retinal and LGN cells are spatially distributed (and follow a retinotopic map) and can thus be described by their locations. We can use the linear filters from chapter 2 to predict their responses over both time and space to arbitrary stimuli:

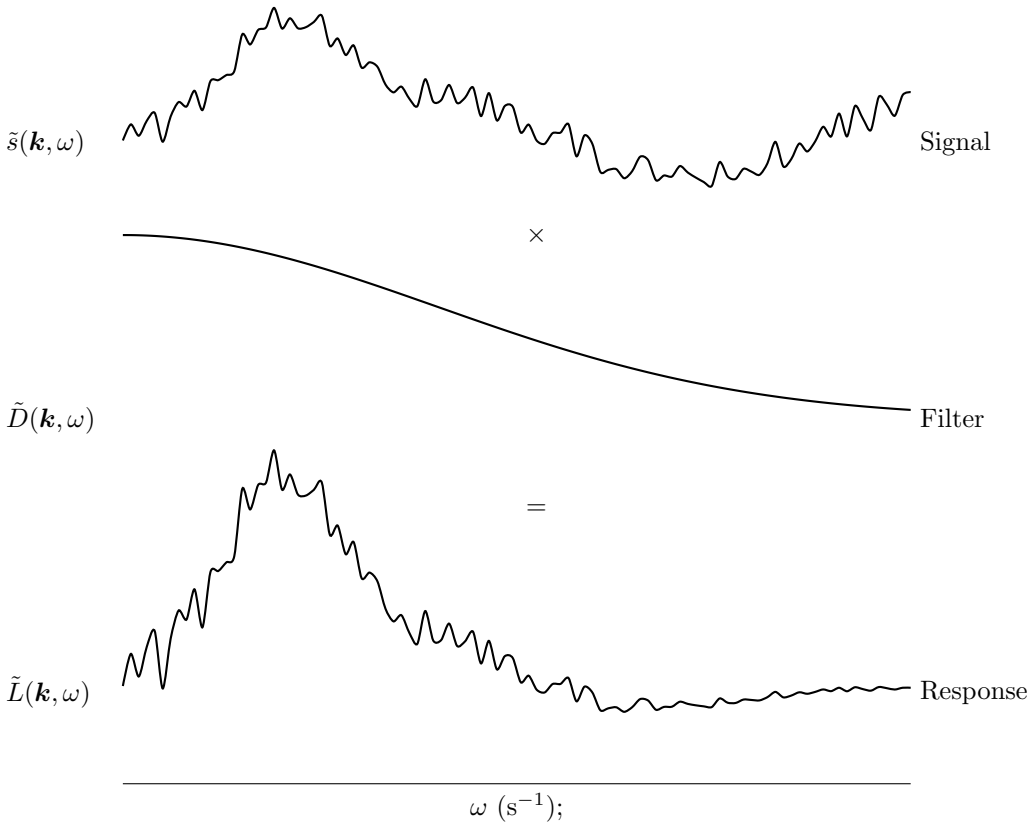
$$L(\boldsymbol{a}, t) = \int_0^\infty d\tau \int d\boldsymbol{x} D(\boldsymbol{x} - \boldsymbol{a}, \tau) s(\boldsymbol{x}, t - \tau)$$

Here, $\boldsymbol{a}$ represents the position of, say, the retinal ganglion cell we are considering, so we have an array of cells that respond differently based on their location to a stimulus that also varies over space and time. Now, just like before, this code will maximise its information about the stimulus if different neural responses are completely uncorrelated and the distributions over their individual responses are entropy maximising. A type of response that approximates these criteria is white noise (this satisifies the first but not necessarily the second criterion). So our approach is to design a filter

such that the cells are responding like white noise. Much like TV static, knowing the response of one cell at one time should tell you nothing about the responses of any other cells at any other times. The book describes how this is done in stages in the visual system: retinal ganglion cells are well described as a white-noise response spatially, but are correlated with themselves in time, and then cells in the LGN further whiten the response in the temporal domain.

We've seen how white noise has a dirac delta autocorrelation function. We haven't discussed the well-known fact that the Fourier transform of white noise is flat. If you are unfamiliar with the Fourier transform, I would recommend reading the appendix, but for the purpose of this discussion, you just need to know that it is a way of representing any signal in terms of the individual frequency components that make it up. White noise consists of equal mixtures of *all* frequencies.

We will also use the convolution theorem to reinterpret what the linear filter above actually does. The neural response is defined as the convolution of the stimulus with a filter, which means we can interpret the function of this filter very straightforwardly by thinking in frequency space. The signal is composed of some distribution over all frequencies, and the filter simply weights these frequencies to produce the response, as shown in the diagram below.



We only plot the frequencies in time, but obviously they exist in space as well. Now, we can measure the spatial and temporal power spectrum of naturally occurring visual stimuli $\tilde{s}(\boldsymbol{k},\omega)$. In order to maximise the information our responses have about the stimuli, it is obvious that we just need to design our filter to invert the natural power spectrum to give a white noise spectrum! This means:

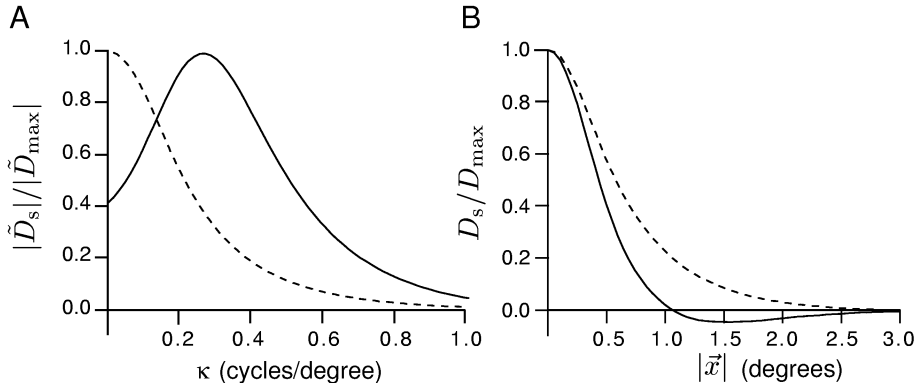
$$\tilde{D}(\boldsymbol{k},\omega) = \frac{\sigma_L}{\tilde{s}(\boldsymbol{k},\omega)}$$

Where σ_L is the white noise power of the linear response. Note this is slightly different to the solution discussed in the book. First of all, I have neglected discussing phases to simplify things (Fourier transforms give you complex numbers!). Additionally, the book uses the Fourier transform of the stimulus autocorrelation function, not the stimulus itself, hence the square root. These are related by the Wiener-Khinchin theorem, so these results are essentially equivalent. I personally find the Fourier transform of an autocorrelation function trickier to conceptualise. So bear in mind that my discussion is aimed at developing intuition, however, if you want to be more formal you should

follow the derivation in the book.

This chapter works through these calculations for naturally occurring stimuli, with some interesting discussions of the results. They also use the additional complication of assuming additive white noise on top of the measured stimulus, which modifies the results slightly, but conceptually our discussion still stands. An interesting result is that they use this information-maximisation approach to derive the retinal ganglion cells' spatial receptive fields, finding results that match experiment well.

Some of these results are actually very interpretable: for example, in the diagram below, the receptive field with the dashed line represents the retinal ganglion cell's receptive field in the presence of high noise, while the solid line represents its response in the presence of low noise. In the high noise case, acheiving an accurate encoding by averaging out the noise over a larger region of space becomes a priority, while in the low noise case, it's more important that neighbouring neurons having non-overlapping receptive fields, hence the centre-surround structure.



5 Chapter 5 - Model Neurons I: Neuroelectronics

In this chapter and the next, we dive into models of single-neuron electrical properties, at an increasing level of complexity. Before doing this, it is worth thinking about why we are modelling single neurons, and the level of depth to which we want to model them.

I begin by referring to David Marr's three levels of analysis: the computational, algorithmic, and implementation levels. Of course, understanding the implementation level is important in its own right, in the same sense that having some idea of how semiconductors and transistors work is important for understanding how digital computers work. However, in the case of digital computers, we know apriori that the fundamental computational units consists of its logic gates, and so we can abstract away these lower-level details and focus on the algorithmic aspects (unless you're a chip designer). In the case of the brain, it is still highly debated what the fundamental computational units are, or in other words, what details we can abstract away and in what contexts, so studying the implementation level is far more pertinent.

That said, it is crucial to understand that lower-level details won't lead to fundamentally new computational capabilities. It can be proved that networks of even the most simple possible single-neuron model, the McCulloch-Pitts neuron, can be both Turing complete and universal function approximators. These are the neurons out of which modern AI systems are constructed. However, caring only that your system is Turing complete and a universal function approximator is a bit like a man with a horse telling Henry Ford he doesn't need to build cars because, in principle, with an infinite amount of time and supplies, they can both travel just as far. In the world of computation, efficiency and speed are key. AI systems leverage the completeness of McCulloch-Pitts neurons with the efficiency of modern chips highly optimised for large linear algebra operations. In the brain, efficiency may well be gained by more complex computations occurring at the single-neuron or sub-neuron levels, and it is our job to figure out what these could be.

5.1 Basic neuron physiology

We begin with a brief summary of the relevant physiological properties of neurons:

- Ion pumps maintain a concentration gradient of different ions across the membrane. The most important ion pump is Na^+/K^+ ATPase, which pumps 3 Na^+ ions out of the cell for every 2 K^+ ions it pumps in, hydrolyzing 1 ATP in the process. This pump alone can be responsible for up to 75% of a neuron's energy expenditure.
- The net effect of all the pumps is such that, at rest, the inside of the neuron has an excess of negative charge, while the outside has an excess of positive charge.
- We measure potential difference from outside to inside, so the resting potential is negative at around -70mV. In total, the potential can vary from around -90mV to +50mV.
- Neuron membranes are made of lipid bilayers, which are impermeable to ions, and therefore act as capacitors.
- The capacitance per unit area is roughly $10nF/mm^2$, and neuronal surface areas vary from around 0.01 to $0.1mm^2$.
- Ion channels are embedded in the membrane, and are highly selective to which ions they allow to pass through. These channels act as resistors in parallel with the capacitor. At rest, the net specific membrance resistance is around $1M\Omega/mm^2$.
- The most important ions are Na^+, K^+, Ca^2+ and Cl^-. These all have different concentrations inside and outside the cell, which means they have thermodynamic diffusive forces acting on them, in addition to electrical forces caused by the potential difference. Typically, the change in concentration gradients is small enough that this diffusive force is roughly constant compared to the electrical force (although there are important exceptions).
- Sodium (Na^+) and potassium (K^+) are responsible for action potential generation. Na^+ is more concentrated outside the cell than inside ($\sim$145mM vs $\sim$12mM). K^+ is more concentrated inside the cell than outside ($\sim$155mM vs $\sim$4mM).
- Calcium (Ca^{2+}) is important for synaptic transmission/vesicle release, among other things. It is far more concentrated outside the cell than inside ($\sim$1-2mM vs $\sim$0.0001mM).
- Chloride (Cl^-) is important for inhibitory synaptic currents and synaptic plasticity, among other things. It is more concentrated outside the cell than inside ($\sim$120mM vs $\sim$4mM).
- We typically model membrane currents as going from inside to outside the cell, so a positive current is a hyperpolarising current, while a negative current is a depolarising current. In contrast, an experimental electrode current is defined as going from outside to inside the cell.

5.2 Membrane currents and the Nernst equation

To construct a point neuron model (which just means we don't worry about morphology), we combine the above elements into an equivalent RC circuit, which allows us to use circuit theory to study how the membrance potential and currents change over time.One of the building blocks of such a model represents the current through ion channels of a single type, and so our first task is to derive the equation relating the membrane potential and current through this channel.

The first step is to find the potential at which the net ion flow through the channel is zero, so that the ionic current is zero. Recall that ions flow through the current due to both electric forces and diffusion due to the concentration gradients. The Nernst equation tells us the potential when the flow due to these two effects is equal and opposite. To solve for this we take an energy based picture. The change in electric potential energy when an ion moves from inside to outside the cell is given by:

$$\Delta U = -zqV$$

This is positive for a positive ion and negative potential. To pass from inside to outside the cell, the ion must have at least this much thermal energy. The energies of ions follow the Boltzmann

distribution (a well established fact within statistical physics), so the probability of an ion having this much energy is given by:

$$P = e^{-\frac{\Delta U}{k_b T}} = e^{\frac{zqV}{k_b T}}$$

Ions will diffuse from outside to inside at a rate proportional to [outside] (their concentration outside the cell), while they will diffuse from inside to outside at a rate proportional to [inside] times the fraction with sufficient thermal energy to cross the potential difference. Setting these equal to each other, and using E to represent the potential at which these balance, gives:

$$[\text{outside}] = [\text{inside}]e^{\frac{zqE}{k_b T}}$$

And solving for E, the equilibrium potential, gives the Nernst equation:

$$E = \frac{k_b T}{zq} \log\left(\frac{[\text{outside}]}{[\text{inside}]}\right)$$

The parameters [outside], [inside], and T are all essentially constant for usual physiological conditions, so we treat the reversal potential as a constant. This can seem strange at first, but it results from the fact that only a relatively small number of ions need to cross the membrane to change the membrane potential, keeping the concentrations essentially constant.

We can calculate the reversal potential for each of the four main ions:

- Sodium: $E_{Na} \sim 50$ to $90mV$
- Potassium: $E_K \sim -90$ to $-70mV$
- Calcium: $E_{Ca} \sim 150mV$
- Chloride: $E_{Cl} \sim -65$ to $-60mV$

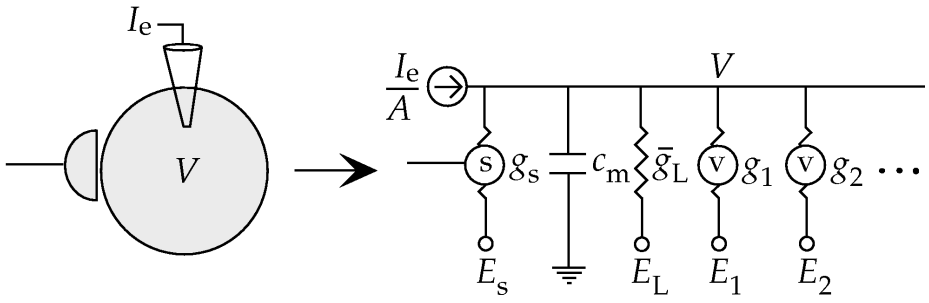
Now consider the current through a channel when the membrane potential is not equal to the reversal potential. This current is given by Ohm's law, however, we have to offset the potential by the reversal potential to account for current driven by the concentration gradient. This gives:

$$i_m = g_i(V - E_i)$$

Where i represents the current through the channel per unit area, g_i is the conductance of channel i per unit area (the inverse of resistance), and E_i is the reversal potential of channel i. The total membrane current is given by summing this up across all ion channels. Note that each ion channel essentially wants to pull the membrane potential towards its reversal potential, and so the resting potential of a neuron is a weighted average of all its ion channel's reversal potentials.

Many ion channels have voltage-dependent conductances, meaning that rather than remaining open with a fixed conductance, they open or close depending on the membrane potential. Biophysically, this is due to conformational changes in the protein structures of the channel - as the potential can essentially tug on the electrostatic features within the channel, since it is embedded in the membrane and thus across the potential difference. If this were not true, then neuron physiology would in fact be incredibly boring - as the dynamics would be entirely linear. Since the sum of linear functions is linear, we can lump all the non-voltage-dependent conductances into a single term, called the leakage term. Its reversal potential and conductance do not represent any single channel, but rather the (weighted) average of all the channels that are not voltage-dependent.

We now construct an equivalent circuit for the neuron, by considering the membrane as a capacitor with specific membrane capacitance c_m, in parallel with resistors representing the various ion channels, as well as an externally injected current source. Note that these ion channels include the voltage-dependent conductances discussed above, the leakage term, and synaptic ion channels with conductances that depend on presynaptic activities (we will return to these later):



Using circuit theory we can write down the equation for the membrane potential:

$$c_m \frac{dV}{dt} = -\bar{g}_L(V - E_L) + \sum_i g_i(V - E_i) + \frac{I_e}{A}$$

The additional voltage or synaptic dependencies not made explicit here. This is an ordinary differential equation and forms the foundation of all neuron models. We now explore two such models: the leaky integrate-and-fire model and the Hodgkin-Huxley model.

5.3 Leaky integrate-and-fire model

The leaky integrate-and-fire model results from making two simplifications. First of all, we ignore all active conductances, considering only the leakage term. Second, we gloss over the precise details of the action potential, replacing it with a dirac-delta spike whenever the membrane reaches a threshold V_{th} and an immediate reset of the membrane potential to V_{reset}. Rewriting our equation above, multiplying by the specific membrane resistance, gives:

$$\tau_m \frac{dV}{dt} = E_L - V + R_m I_e$$

Where τ_m is the membrane time constant $c_m r_m$. Before adding spikes, we discuss solutions to this equation as is. In the case of a constant input current, the solution is simply an exponential relaxation to the steady state value $E_L + R_m I_e$, with a time constant τ_m:

$$V(t) = E_L + R_m I_e + (V(0) - E_L - R_m I_e)e^{-t/\tau_m}$$

In the case of a time-varying input current $I_e(t)$ (which we often use to model synaptic inputs), this equation is simply a low-pass filter of the input signal $I_e(t)$, offset by E_L. If you are not that familiar with simple ODEs such as this it is informative to go through the derivation of this. First of all, we rearrange the equation slightly and multiply by an integrating factor $\exp \frac{t}{\tau_m}$:

$$\left(\frac{dV}{dt} + \frac{V}{\tau_m}\right) e^{\frac{t}{\tau_m}} = \frac{1}{\tau_m} \left(E_L + R_m I_e(t)\right) e^{\frac{t}{\tau_m}}$$

In this form, the left-hand side simplifies:

$$\frac{d}{dt}\left(V(t) e^{\frac{t}{\tau_m}}\right) = \left(E_L + R_m I_e(t)\right) e^{\frac{t}{\tau_m}}$$

Integrating both sides from $-\infty$ to t, and simplifying both the left-hand side and constant term on the right gives:

$$V(t) = E_L + R_m \int_{-\infty}^{0} e^{-\frac{t'}{\tau_m}} I_e(t - t')dt'$$

If you look at the integral term, you see we have a weighted average of the input current over its history, multiplied by an exponentially decaying kernel. This is a low-pass filter: it roughly tracks the input current, but smooths out high frequency fluctuations. A Fourier analysis of this solution shows that its exactly equal to removing the higher frequency components of $I_e(t)$.

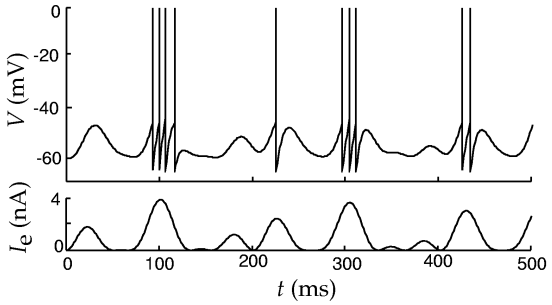
We can now add spikes whenever the membrane potential reaches V_{th}. Unlike in the book, we will do this explicitly by using a dirac delta:

$$\tau_m \frac{dV}{dt} = E_L - V + R_m I_e + \sum_i^N \delta(t - t_k)(V_{reset} - V_{th})$$

With spikes times t_k and the complete spike train $S(t)$ given by:

$$t_k = \arg_t (V(t) = V_{th})$$
$$S(t) = \sum_k^N \delta(t - t_k)$$

As shown in the diagram below, this is exactly the low-pass filter, with instantaneous spikes shown at the spike times:



At this point, there is one informative exercise we can do, which is derive the firing rate of the LIF neuron as a function of a constant input current. This is straightforward to do: assuming $V(0) = E_L$, we can use our constant-input solution to find the time T of the first spike:

$$V_{th} = E_L + R_m I_e - R_m I_e e^{-T/\tau_m}$$

Rearranging for T gives:

$$T = \tau_m \log\left(\frac{R_m I_e}{R_m I_e + E_L - V_{th}}\right)$$

The firing rate is simply the inverse of this:

$$r = \frac{1}{\tau_m \log\left(\frac{R_m I_e}{R_m I_e + E_L - V_{th}}\right)}$$

This result is physically meaningful only when $R_m I_e + E_L > V_{th}$, otherwise the neuron will never spike. Note my derivation here is slightly simpler than that used in the book in that I set the reset potential to the resting potential, which is commonly done (and the alternative adds nothing new conceptually). We now show that this becomes approximately linear in the case of large input currents. First, we rewrite:

$$r = \frac{1}{\tau_m \log\left(1 - \frac{E_L - V_{th}}{R_m I_e + E_L - V_{th}}\right)}$$

And use the Taylor expansion of the logarithm: $\log(1 + x) \approx x$ for small x. This gives:

$$r \approx \frac{1}{\tau_m}\left(1 + \frac{R_m I_e}{V_{th} - E_L}\right) \text{ if } R_m I_e \gg V_{th} - E_L$$

5.4 Hodgkin-Huxley model

5.5 Synaptic conductances

5.6 A brief historical note and further reading

This book is over 20 years old and, obviously, there have been a huge number of developments in the field since then. The topics we've covered here - the LIF neuron, Hodgkin-Huxley neuron, and

synaptic conductances, remain incredibly relevant as a foundation. There are different directions in which one can expand upon these models. On the one hand, you can incorporate increasing levels of detail to capture neuron physiology better. This includes some of the work done in the next chapter on modelling neuron morphology with cable theory, and incorporating more ion channels such as in the Connor-Stevens model. This is an almost infinite rabbit hole to go down, and so I will omit these extensions in the notes.

On the other hand, there is an increasing emphasis in the field on working on models that can capture the essential features of what a neuron computes, in a manner that can be very efficiently simulated, and that can be easily incorporated into larger networks. These models often don't concern themselves as much with exact biological realism, but envision neurons as more abstract computational units, and work in this field is often much more inspired by developments in machine learning. I recommend here some further reading down these lines. I do not intend this to be in any way opinionated on how much detail *should* be included in a model, as this depends on your modelling context, and opinion still varies massively on how much detail is necessary to understand higher-level computational properties of networks.

- **Exponential integrate-and-fire model**: Incorporates more detail of membrane potentials around the threshold with an exponential non-linearity.
- **Izhikevich model**: A two-dimensional model that can capture a wide variety of spiking behaviours present in Hodgkin-Huxley neurons while being more efficient to simulate.
- **Adaptive exponential integrate-and-fire models**: A further extension of the exponential integrate-and-fire model that incorporates spike frequency adaptation.
- **Escape noise models**: These are stochastic models that can better approximate the noisiness of real spike generation.
- **Multicompartment models**: I include a treatment of multicompartment models in the next chapter because there is fascinating work suggesting compartments may be necessary to understand how, for example, neurons compute gradient approximations to implement first-order learning rules, great work on dendritic integration, and much more.
- **Surrogate gradients**: Training networks of LIF-like neurons (called spiking neural networks (SNNs)) to do useful things is incredibly difficult. Surrogate gradients are now the technique of choice for training them, bringing the power of backpropagation to SNNs.
- **Synaptic models**: There's a wealth of models that use LIF-like neurons with more sophisticated synaptic dynamics, to include effects such as short-term plasticity, intelligent weight updates, and far more.