Neural Computation
Practical 6: Information in spike trains

2004/5 version by Mark van Rossum
2018 version by Peggy Series, Matthias Hennig, and Theoklitos Amvrosiadis

November 7, 2018

1 Aims

- Simulate a Poisson spike train
- Familiarise with the concepts of joint, conditional, and marginal probability
- Explore basic principles of Neural Information Theory

2 Theoretical Background

2.1 Probability Theory

The basics of probability theory are crucial for the understanding of Information Theory. A full coverage of the topic cannot be given here for obvious reasons, so you are encouraged to have a brief look at the fundamental concepts of probabilities before starting this practical, if you don't feel very confident. Just the very basics:

- For two discrete random variables, x, y, the **joint** probability distribution, $P(X, Y)$, represents the probability that $x = X$ **AND** $y = Y$[1], for any X, Y. If the variables are independent, then $P(X, Y) = P(X)P(Y)$.
- The **conditional** probabilities $P(X|Y)$ and $P(Y|X)$, represent the probabilities that $x = X$ *given that* $y = Y$, and vice versa. For discrete random variables, it is: $P(X|Y)P(Y) = P(X, Y) = P(Y|X)P(X)$.
- $P(X)$ and $P(Y)$ are called the **marginal** probabilty distributions of the random variables, and they represent what values each variable takes *independent* of what the other is doing. Hence, we can get the marginal probability distribution for one variable by summing the joint distribution with respect to the other variable.[2] For example, $P(X) = \sum_y P(X, Y)$.[3]

The above are easily seen with 2-way tables for two discrete variables, but can be generalized for multivariate distributions and continuous variables as well.

2.2 Poisson spike train

For a Poisson-spiking neuron the probability of firing a spike in a time interval (time bin) Δt is given by $p = \rho_0 \Delta t \in \{0...1\}$, where ρ_0 is the firing rate of the neuron. When time bin gets arbitrarily small, this probability vanishes. However, by considering a time bin of approximately $1ms$, we would have a maximum of one spike/bin for most neurons. (Why is that? Remember the earlier practicals!)

[1] Can also be expressed as $P(x = X \cap y = Y)$
[2] The second variable is then "marginalized", since it is not taken into account any more.
[3] And given the equation for the conditional probabilities, $P(X) = \sum_y P(X, Y) = \sum_y P(X|Y)P(Y)$

For simplicity, we will not deal here with firing rates and interval durations. We will have a standard time bin (think of it as having a duration of $1ms$, if you want) and the stimulus s that we will apply will directly *induce* a probabilty $s = p = \{0...1\}$ of observing a spike in each time bin.

Let's consider the case where we look at four consecutive time bins and we observe the firing pattern (1, 0, 0, 1). We can think of this response pattern as a "code word" the neuron uses to communicate to downstream targets. We play the role of the downstream neuron, reading the output of the neuron, one word (4-bin-long in this case) at a time. Just to put things in perspective, a neuron in the primary visual cortex (V1) has no direct access to the visual world. All it knows are what neurons in the lateral geniculate nucleus (LGN) communicate, which in turn only know what retinal ganglion cells (RGCs) are passing on, and so on..

3 Setting up the simulation in MATLAB

In the simulations to follow, we will look at the firing pattern of one neuron, in a certain number of trials, in response to stimulation. Thus, you will need a variable for the number of bins we will be looking at, *nbins*, one for the number of trials which we will simulate, *ntrials*, the number of distinct stimuli, *nstims,* and the value of each stimulus, s (remember that it corresponds to the probability of observing a single spike in a bin).

- To generate the Poisson spike train you can make use of the *rand* function*,* which gives you an array of uniformely distributed random numbers from 0 to 1. How would you then turn this into a binary output, depending on the stimulus *s?* There are at least 2 ways, one using the function *floor,* and another using a relational operator.[4]

- Next, we need to assign a unique label to each possible "word" of the neuron. (How many are there for $nbins = 4$? How do you calculate for any *nbins* value?) An easy way to do this would be to turn the each word into a number in the base-2 system. So, for the above pattern (1, 0 , 0, 1), the label would be $1 * 2^0 + 0 * 2^1 + 0 * 2^2 + 1 * 2^3 = 9$[5]. This gives us a one-to-one mapping of spike patterns to labels which will be very useful in the next steps. You can create this "base" once in the start and then use it when necessary. Think about how you can create it for any value of *nbins*.[6]

- We will also need to store the joint probability, $P(r, s)$, where r are the responses (taken as whole words) of the neuron, and s are the different stimulus values we will provide. How many elements will this array have (as a function of *nbins* and *nstims*)? You can create initialize it with *zeros*, since it will make things faster.

4 Starting the simulation

As mentioned above, we will explore the responses of our neuron to a number of different stimuli, *nstims,* with intensities ranging from 0 to 1, in a certain number of trials, *ntrials.* Let *nstims = 50* and *ntrials = 100* to begin with.

- First, calculate the stimulus intensity (=spike probability) values.[7]

- Then, you will need to calculate the response of the neuron in *each trial* for *each stimulus intensity.*[8] It would be best to store these responses together in a 3D-matrix, although there are certainly other, maybe better, ways. One dimension would be *nbins,* one *ntrials,* and one *nstims.*

[4] For example, *floor(rand(nbins, ntrials) + s),* or *rand(nbins, ntrials)>s.* Do these work in the same way exactly?

[5] How would you perform this operation in MATLAB, which specializes in matrix operations?

[6] *base = zeros(nbins, 1);*
for ii = 1:nbins
base(ii) = 2^(ii-1);
end

[7] Divide the interval $0 \longrightarrow 1$ into *nstims* parts.

[8] ..which means, loops.

- Now, it's time to make use of our "encoding". For each trial, calculate the output of the neuron as a word (remember the **base** we created above).
- For each given stimulus intensity, first count how many times each word appears (not manually!). Here is the tricky part. A way to do this, would be to use the value of each word as an *index* into the $P(r, s)$ matrix you created in Section 3. Every time you encounter the word, you increment the counter at the corresponding index by 1. Be careful with indexing, especially if you are used to other programming languages, since MATLAB indexing starts from 1.[9]
- Then, turn this into a probability. (How do we turn counts into probabilities?) You should now have the joint probability distribution $P(r, s)$.

5 Mutual Information

Great, that's the easy part done. On to the fun part.

Mutual information between two random variables represents the amount of information that we gain about one by observing the other. Equivalently, it measures the information that is shared between the two variables. If the two variables are *independent*, then their mutual information is 0, since, by definition, knowing the value of one does not provide any information about the other.

Formally, the mutual information for discrete variables is expressed as:

$I(x, y) = \sum_y \sum_x p(x, y) log_2 \frac{p(x,y)}{p(x)p(y)}$

This is equivalent to $I(x, y) = H(y) - H_{noise} = H(y) - H(y|x)$. If from the information present in the output, y, we subtract what is *not dependent* on the input x (aka, noise), we are left with the information in the output that *is* dependent on the input, and thus the *mutual* information.

In our case, these equations would take the form:

$I_m = \sum_s \sum_r P(r, s) log_2 \frac{P(r,s)}{P(r)P(s)}$

- So, a good first step to calculate the mutual information would be to find $P(r)$ and $P(s)$. Remember, these are the marginal probabilities. The way to calculate them is shown in Section 2. Remember also that you can use *sum* across different directions of a matrix.[10]
- Make sure that you have calculated the marginal probabilities correctly. An easy way to check this is that it should be, $\sum_r P(r) = \sum_s P(s) = \sum_{r,s} P(r, s) = 1$.
- Calculate the mutual information for *each* stimulus intensity, looping over word patterns. Then, calculate the total mutual information for all stimulus intensities.
- What units is mutual information (Information, in general) measured in? Calculate the information *per spike.*

[9] I realize this might be too cryptic for a lot of people, so I include part of the code. This is not commented on purpose and there are obviously parts that you would need to have written above for it to work. Try to work it out for yourselves!

```
for istim = 1:nstim
for itrial = 1:ntrials
word = spikes(itrial, :, istim);
resp(itrial, istim) = word*base + 1;
Prs(istim, resp(itrial, istim)) = Prs(istim, resp(itrial, istim)) + 1;
end
end
```

[10] You can also work with conditional probabilities if you wish to, modifying the equation accordingly.

6 Exploring Neural Information Theory

- What is the input entropy? What is the output entropy? Remember that, in general, $H(x) = -\sum_i p(x_i) log_2 p(x_i)$
- Increase the number of bins. What do you expect to happen? Plot the mutual information as a function of number of bins. What is the shape you observe? Are there any constraints on this function?[11]
- Explore the effects of the *ntrials* variable. Make another plot of mutual information as its function.
- Change the stimulus intensity distribution and look at the mutual information again.

[11] Hint: Shannon's source coding theorem