# Convolution and correlation

## Sebastian Seung

## 9.29 Lecture 2: February 6, 2003

In this lecture, we'll learn about two mathematical operations that are commonly used in signal processing, *convolution* and *correlation*. The convolution is used to linearly filter a signal, for example to smooth a spike train to estimate probability of firing. The correlation is used to characterize the statistical dependencies between two signals.

A few words about the big picture. The previous lecture discussed how to construct a linear model relating firing rate and stimulus at a single time. In the next lecture, convolution and correlation will be used to construct linear models that relate neural firing rate to a stimulus, but at multiple times.

# 1 Convolution

Let's consider two time series, $g_i$ and $h_i$, where the index $i$ runs from $-\infty$ to $\infty$. The convolution of these two time series is defined as

$$(g * h)_i = \sum_{j=-\infty}^{\infty} g_{i-j} h_j \tag{1}$$

This definition is applicable to time series of infinite length. If $g$ and $h$ are finite, they can be extended to infinite length by adding zeros at both ends. After this trick, called zero padding, the definition in Eq. (1) becomes applicable. For example, the sum in Eq. (1) becomes

$$(g * h)_i = \sum_{j=0}^{n-1} g_{i-j} h_j \tag{2}$$

for the finite time series $h_0, \dots, h_{n-1}$.

**Exercise 1** *Convolution is commutative and associative. Prove that $g * h = h * g$ and $f * (g * h) = (f * g) * h$.*

**Exercise 2** *Convolution is distributive over addition. Prove that $(g_1 + g_2) * h = g_1 * h + g_2 * h$. This means that filtering a signal via convolution is a linear operation.*

Although $g$ and $h$ are treated symmetrically by the convolution, they generally have very different natures. Typically, one is a signal that goes on indefinitely in time. The

1

other is concentrated near time zero, and is called a filter. The output of the convolution is also a signal, a filtered version of the input signal.

Note that filtering a signal via convolution is a linear operation. This is an important property, because it simplifies the mathematics. There are also nonlinear methods of filtering, but they involve more technical difficulties. Because of time limitations, this class will cover linear filters only. Accordingly, we will discuss only neurobiological examples for which linear models work well. But these examples are exceptions to the rule that most everything in biology is nonlinear. Don't jump to the conclusion that linear models are always sufficient.

In Eq. (2), we chose $h_i$ to be zero for all negative $i$. This is called a *causal filter*, because $g * h$ is affected by $h$ in the present and past, but not in the future. In some contexts, the causality constraint is not important, and one can take $h_{-M}, \ldots, h_M$ to be nonzero, for example.

Formulas are nice and compact, but now let's draw some diagrams to see how convolution works. To take a concrete example, assume a causal filter $(h_0, \ldots, h_{n-1})$. Then the $i$th component of the convolution $(g * h)_i$ involves aligning $g$ and $h$ this way:

$$
\begin{array}{ccccccccccc}
\cdots & g_{i-m-1} & g_{i-m} & g_{i-m+1} & \cdots & g_{i-2} & g_{i-1} & g_i & g_{i+1} & g_{i+2} & \cdots \\
\cdots & 0 & 0 & h_{m-1} & \cdots & h_2 & h_1 & h_0 & 0 & 0 & \cdots
\end{array}
$$

In words, $(g * h)_i$ is computed by looking at the signal $g$ through a window of length $m$ starting at time $i$ and extending back to time $i - m + 1$. The weighted sum of the signals in the window is taken, using the coefficients given by $h$.

If the signal $g$ has a finite length, then the diagram looks different when the window sticks out over the edges. Consider a signal $g_0, \ldots, g_{m-1}$. Let's consider the two extreme cases where the window includes only one time bin in the signal. One extreme is $(g * h)_0$, which can be visualized as

$$
\begin{array}{ccccccccc}
\cdots & 0 & \cdots & 0 & g_0 & g_1 & \cdots & g_{n-1} & 0 & \cdots \\
\cdots & h_{m-1} & \cdots & h_1 & h_0 & 0 & 0 & 0 & 0 & \cdots
\end{array}
$$

The other extreme is $(g * h)_{m+n-2}$, which can be visualized as

$$
\begin{array}{ccccccccc}
\cdots & g_0 & g_1 & \cdots & g_{n-1} & \cdots & 0 & 0 & 0 & \cdots \\
\cdots & 0 & 0 & \cdots & h_{m-1} & \cdots & h_1 & h_0 & 0 & \cdots
\end{array}
$$

Therefore $g * h$ has $m + n - 1$ nonvanishing components.

## 2 Using the MATLAB `conv` function

If $g_0, g_1, \ldots, g_{M-1}$ and $h_0, h_1 \ldots, h_{N-1}$ are given as arguments to the `conv` function, then the output is $f_0, f_1, \ldots, f_{M+N-2}$, where $f = g * h$. More generally, if $g_{M_1}, \ldots, g_{M_2}$ and $h_{N_1}, \ldots, h_{N_2}$ are given as arguments to the `conv` function, then the output is $f_{M_1+N_1}, \ldots, f_{M_2+N_2}$. In other words, shifting either $g$ or $h$ in time is equivalent to shifting $g * h$ in time.

For example, suppose that $g$ is a signal, and $h$ represents an acausal filter, with $N_1 < 0$ and $N_2 > 0$. Then the first element of $f$ returned by `conv` is $f_{M_1+N_1}$, and the

last is $f_{M_2+N_2}$. So discarding the first $|N_1|$ and last $N_2$ elements of $f$ leaves us with $f_{M_1}, \ldots, f_{M_2}$. This is time-aligned with the signal $g_{M_1}, \ldots, g_{M_2}$, and has the same length.

Another motivation for discarding elements at the beginning and end is that they may be corrupted by edge effects. If you are really worried about edge effects, you may have to discard even more elements, which will leave $f$ shorter than $g$.

## 3 Firing rate

Consider a spike train $\rho_1, \ldots, \rho_N$. One estimate of the probability of firing is

$$ p = \frac{1}{N} \sum_i \rho_i \tag{3} $$

This estimate is satisfactory, as long as it makes sense to describe the whole spike train by a single probability that does not vary with time. This is an assumption of statistical *stationarity*.

More commonly, it's a better model to assume that the probability varies slowly with time (is nonstationary). Then it's better to apply something like Eq. (3) to small segments of the spike train, rather than to the whole spike train. For example, the formula

$$ p_i = (\rho_{i+1} + \rho_i + \rho_{i-1})/3 \tag{4} $$

estimates the probability at time $i$ by counting the number of spikes in three time bins, and then dividing by three. In the first problem set, you were instructed to smooth the spike train like this, but to use a much wider window. In general, choosing the size of window involves a tradeoff. A larger window minimizes the effects of statistical sampling error (like flipping a coin many times to more accurately determine its probability of coming up heads). But a larger window also reduces the ability to follow more rapid changes in the probability as a function of time.

Note that Eq. (4) isn't to be trusted near the edges of the signal, as the filter operates on the zeros that surround the signal.

There are other methods for estimating probability of firing, many of which can be expressed in the convolutional form,

$$ p_i = \sum_j \rho_{i-j} w_j $$

where $w$ satisfies the constraint $\sum_j w_j = 1$. According to this formula, $p_i$ is the weighted average of $\rho_i$ and its neighbors, so that $0 \leq p_i \leq 1$. Closely related is the firing rate per unit time,

$$ \nu_i = \frac{p_i}{\Delta t} $$

where $\Delta t$ is the length of a time bin, or sampling interval. Probabilistic models of neural activity will be treated more formally in a later lecture, and we'll example concepts like firing rate more critically.

There are many different ways to choose $w$, depending on the particulars of the application. Previously we chose $w$ be of length $n$, with nonzero values equal to $1/n$. This is sometimes called a "boxcar" filter. MATLAB comes with a lot of other filter shapes. Try typing `help bartlett`, and you'll find more information about the Bartlett and other types of windows that are good for smoothing. Depending on the context, you might want a causal or a noncausal filter for estimating probability of firing.

Another option is to choose the kernel to be a decaying exponential,

$$w_j = \begin{cases} 0, & j < 0 \\ \gamma(1-\gamma)^j, & j \geq 0 \end{cases}$$

This is causal, but has infinite duration.

**Exercise 3** *Prove that the exponential filter is equivalent to*

$$p_i = (1-\gamma)p_{i-1} + \gamma\rho_i$$

# 4   Impulse response

Consider the signal consisting of a single impulse at time zero,

$$\delta_j = \begin{cases} 1, & j = 0 \\ 0, & j \neq 0 \end{cases}$$

The convolution of this signal with a filter $h$ is

$$(\delta * h)_i = \sum_k \delta_{j-k} h_k = h_j$$

which is just the filter $h$ again. In other words $h$, is the response of the filter to an impulse, or the *impulse response* function. If the impulse is displaced from time 0 to time $i$, then the result of the convolution is the filter $h$, displaced by $i$ time steps.

A spike train is just a superposition of impulses at different times. Therefore, convolving a spike train with a filter gives a superposition of filters at different times.

Elsewhere you may have seen the "Kronecker delta" notation $\delta_{ij}$, which is equivalent to $\delta_{i-j}$. The Kronecker delta is just the identity matrix, since it is equal to one only for the diagonal elements $i = j$. You can think about convolution with $\delta_j$ as multiplication by the identity matrix $\delta_{ij}$. More generally, the following exercise shows that convolution is equivalent to multiplication of a matrix and a vector.

**Exercise 4** *Matrix form of convolution. Show that the convolution of $g_0, g_1, g_2$ and $h_0, h_1, h_2$ can be written as*

$$g * h = Gh$$

*where the matrix $G$ is defined by*

$$G = \begin{pmatrix} g_0 & 0 & 0 \\ g_1 & g_0 & 0 \\ g_2 & g_1 & g_0 \\ 0 & g_2 & g_1 \\ 0 & 0 & g_2 \end{pmatrix} \tag{5}$$

4

*and $g * h$ and $h$ are treated as column vectors.*

**Exercise 5** *Each column of $G$ is the same time series, but shifted by a different amount. Use the MATLAB function* `convmtx` *to create matrices like $G$ from time series like $g$. This function is found in the Signal Processing Toolbox.*

*If you don't have this toolbox installed, you can make use of the fact that Eq. (5) is a Toeplitz matrix, and can be constructed by giving its first column and first row to the* `toeplitz` *command in MATLAB.*

**Exercise 6** *Convolution as polynomial multiplication. If the second degree polynomials $g_0 + g_1 z + g_2 z^2$ and $h_0 + h_1 z + h_2 z^2$ are multiplied together, the result is a fourth degree polynomial. Let's call this polynomial $f_0 + f_1 z + f_2 z^2 + f_3 z^3 + f_4 z^4$. Show that this is equivalent to $f = g * h$.*

# 5  Correlation

The correlation of two time series is

$$\mathrm{Corr}[g, h]_j = \sum_{i=-\infty}^{\infty} g_i h_{i+j} \tag{6}$$

The case $j = 0$ corresponds to the correlation that was defined in the first lecture. The difference here is that $g$ and $h$ are correlated at times separated by the lag $j$. Note that is the convention followed by Dayan and Abbott. Some other books, like *Numerical Recipes*, call the above sum $\mathrm{Corr}[h, g]_j$. This can be confusing.

As with the convolution, this definition can be applied to finite time series by using zero padding. Note that $\mathrm{Corr}[g, h]_j = \mathrm{Corr}[h, g]_{-j}$, so that the correlation operation is not commutative. Typically, the correlation is applied to two signals, while its output is concentrated near zero.

The zero lag case looks like

$$
\begin{array}{ccccccc}
\cdots & 0 & g_1 & g_2 & \cdots & g_n & 0 & \cdots \\
\cdots & 0 & h_1 & h_2 & \cdots & h_n & 0 & \cdots
\end{array}
$$

and the other lags correspond to sliding $h$ right or left.

The autocorrelation is a special case of the correlation, with $g = h$. If $g \neq h$, the correlation is sometimes called the crosscorrelation to distinguish it from the autocorrelation. In the first lecture, we distinguished between correlation and covariance. The covariance was defined as the correlation with the means subtracted out. Similarly, the cross-covariance can be defined as the correlation left between two time series after subtracting out the means. The auto-covariance is a special case. The command `xcov` can be used for this purpose.

**Exercise 7** *Prove that the autocorrelation is the same for equal and opposite time lags $\pm j$.*

# 6 Using the MATLAB `xcorr` function

If $g$ and $h$ are $n$-dimensional vectors, then the MATLAB command `xcorr(g,h)` returns a $2n - 1$ dimensional vector, corresponding to the lags $j = -(n - 1)$ to $n + 1$. Lags beyond this range are not included, as the correlation trivially vanishes. The $n$th element of the result corresponds to zero lag.

One irritation is that MATLAB 5 and 6 follow different conventions. MATLAB 5 calls Eq. (6) `xcorr(g,h)`, while MATLAB 6 calls it `xcorr(h,g)`.

A maximum lag can also be given as an argument, `xcorr(g,h,maxlag)`, to restrict the range of lags computed to `-maxlag` to `maxlag`. Then the `maxlag+1` element corresponds to zero lag.

The default is the unnormalized correlation given above, but there is also a normalized version that looks like

$$Q_j^{xy} = \frac{1}{m} \sum_{i=1}^{m} x_i y_{i+j}$$

To compensate for boundary effects, the form

$$Q_j^{xy} = \frac{1}{m - |j|} \sum_{i=1}^{m} x_i y_{i+j}$$

is sometimes preferred. Both forms can be obtained through the appropriate options to the `xcorr` command.

# 7 Two examples

The autocorrelation of a sine wave. The period should be evident.

The autocorrelation of Gaussian random noise. There is a peak at zero, while the rest is small. As the length of the signal goes to infinity, the ratio of the sides to the center vanishes. If the autocorrelation of a signal vanishes, except at lag zero, we'll call it *white noise*. The origin of this term will become clear when we study Fourier analysis.

# 8 Spike-triggered average

I define the spike-triggered average as

$$C_j = \frac{\sum_i \rho_i s_{i+j}}{\sum_k \rho_k}$$

This is the cross-correlation of the spike train and the stimulus, normalized by the number of spikes. Note that Dayan and Abbott define the spike-triggered average with the opposite sign in the time lag. However, they also graph it with the time axis reversed, so their graph looks the same as mine would!

You can think about this as taking a snapshot of the stimulus triggered by each spike, and then averaging the snapshots together. This gives an idea of what stimulus waveform is most effective at causing a spike.