SYDE 556/750
Simulating Neurobiological Systems
Lecture 4: Temporal Representation

Andreas Stöckel

Based on lecture notes by
Chris Eliasmith and Terrence C. Stewart

January 23 & 28, 2020

UNIVERSITY OF WATERLOO

Accompanying Readings: Chapter 4 of Neural Engineering

Contents

1   Introduction

Note: We set out to build a theory describing what "the neural code" may be. In Lecture 2 we discussed individual neuron models, and roughly modelled their behaviour in terms of a rate approximation $G[J]$. We continued in Lecture 3 to think about "representation" in groups of neurons, i.e., what the *activity* of individual neurons represents in terms of (external) stimuli. While this brings us closer to a theory of the "neural code", one important notion has been missing so far: time.

We discussed how neural populations could be modelled as representing vectorial quantities. In particular, we assumed that each neuron in a population essentially computes a spike rate $G[J]$ in spikes per second, given an input current J. As we saw when we were discussing the LIF neuron model, this is not the case in biology. Neurons are dynamical systems that receive stimuli over time and output a sequence of spikes.

The output of a biologically plausible neuron model is also called a *spike train*. Given an input current over time, the i-th neuron in a neural population produces a series of output spikes at time points $t_i^1, t_i^2, \ldots$, and so on. We model a spike train as a continuous function over time by superimposing a set of Dirac-δ functions

$$a_i(t) = \sum_{k=1}^{\infty} \delta\big(t - t_i^k\big).$$

Note: As detailed in the notes for Lecture 2, the Dirac-δ function is defined as the function $\delta(t)$ that is zero for all points $t \neq 0$, but has a definite integral from $-\infty$ to ∞ of one.

As before, our goal is to decode ("read out") the value represented in the neural population. However, in contrast to what we did before, we would like to decode the value represented in the population at *any time* t. We will solve this problem in three steps: first, we explore what happens if we use exactly the same technique we used before on the spike trains produced by a neural population. Then, we try to remove noise in the estimate by computing an optimal filter. Lastly, we consider what the biological correlate of such a filter could be.

2   Solving for a Spike Train Decoder

Our first attempt at decoding a neural representation is similar to what we did before – we gather a bunch of training data, consisting of a time-series of neural activity $\boldsymbol{a}(t)$ and a represented value $\mathbf{x}(t)$. We arrange discrete samples of these activities in matrices $\mathbf{A}$ and $\mathbf{X}$ and compute a decoder $\mathbf{D}$. Then, we can decode the represented value according to the decoding equation $\hat{\mathbf{x}}(t) = \mathbf{D}\boldsymbol{a}(t)$.

2.1   Discretizing Spike Trains

In the previous lecture, we discussed computing a decoder $\mathbf{D}$ by choosing a random set of N samples $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \subset \mathbb{X}$, forming a matrix $\mathbf{X} \in \mathbb{R}^{d \times N}$, as well as recording the associated nonnegative population activities $\{\mathbf{a}(\mathbf{x}_1), \ldots, \mathbf{a}(\mathbf{x}_N)\} \subset \mathbb{R}^n$, forming a matrix $\mathbf{A} \in \mathbb{R}^{n \times N}$, where n is the number of neurons in the population.

Now, the question is how to gather such training data in the time-continuous case. One procedure that could accomplish this is to generate a random signal, $\mathbf{x}(t)$, feeding this signal into the neuron population, and recording the neural activities. In other words, for each neuron i in the population, we have the encoding equation

$$J_i(t) = \alpha_i \langle \mathbf{e}_i, \mathbf{x}(t) \rangle + J_i^{\text{bias}},$$

$$a_i(t) = \sum_{k=1}^{\infty} \delta(t - t_i^k),$$

where t_i^k is the time of the k-th spike produced by our neuron model for an input current $J_i(t)$. In case we chose the LIF neuron model, we would for example use $J_i(t)$ as an input in the following sub-threshold differential equation

$$\frac{\mathrm{d}}{\mathrm{d}t} v_i(t) = -\frac{1}{\tau_{\text{RC}}} \big(v_i(t) - J_i(t)\big), \qquad\qquad \text{if } v_i(t) < 1,$$

with the appropriate super-threshold behaviour following spike production – whenever $v_i(t)$ reaches the threshold potential $v_{\text{th}} = 1$ (assume that this happens at a time t_i^k, where k is an incrementing spike index), we clamp the membrane potential to the reset potential $v_{\text{reset}} = 0$ for τ_{ref} seconds

$$v(t) \leftarrow 0, \qquad\qquad \text{if } t > t_i^k \text{ and } t \leq t_i^k + \tau_{\text{ref}}, \text{ for all spike times } t_i^k.$$

To summarize, we have two time-continuous functions: $\mathbf{x}(t)$, our randomly created input function, and $\mathbf{a}(t)$, the spike trains produced by our neurons for the given represented values.

Now, how do we assemble our matrices $\mathbf{X}$ and $\mathbf{A}$? Note that we have described $\mathbf{x}(t)$ and $\mathbf{a}(t)$ in continuous time, yet these matrices need to be constructed out of N discrete values. The answer is that we simply discretise our function into short time-slices of width Δt, starting at a time zero, and stopping at a time T. We then have $N = \lfloor T/\Delta t \rfloor$ time slices, and each of these time slices can be interpreted as one of N samples for our matrix.

✎ **Note:** *Discrete Dirac-δ.* We discuss generating a discrete random input function $\mathbf{x}(t)$ below. However, it is unclear how to discretise the population response $\mathbf{a}(t)$, since each individual neural response $a_i(t)$ is a sum of Dirac-δ functions. Luckily, this can be easily resolved by recalling the two important properties of the Dirac-δ listed above: being zero at all except for one point, and the integral being one. If we want to retain these properties in the discrete case, this means that the Dirac-δ is zero for all sample points except for one sample point which we have to set to Δt^{-1} to get an area-under-the-curve of one when we

(a) Zero mean white white noise

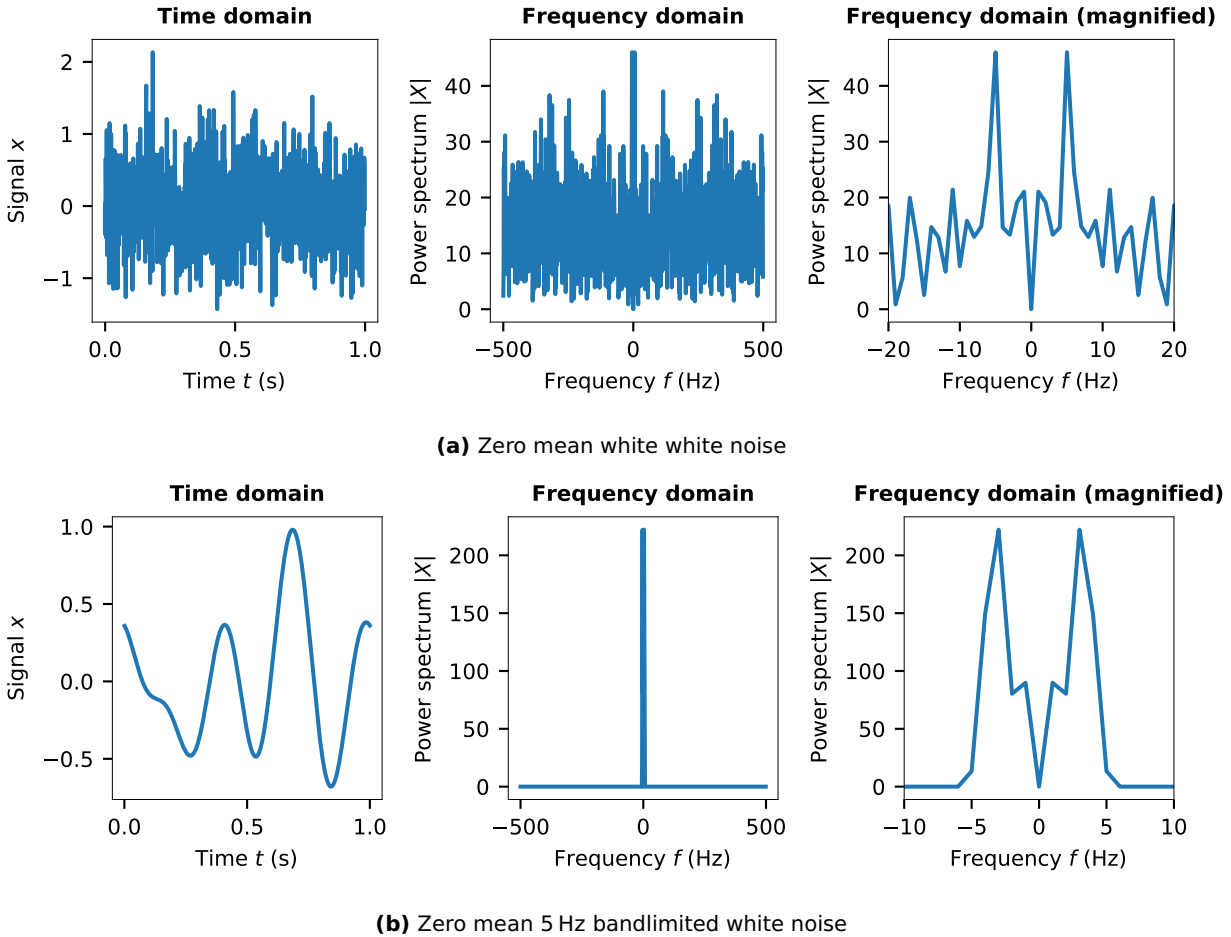**(b)** Zero mean 5 Hz bandlimited white noise

Figure 1: Comparison between white noise and bandlimited white noise. Discrete signals generated with a time step of $\Delta t = 1$ ms. ⌨ *Code*

discretise the integral. Assuming we have a spike at time t' with $0 < t' < T$, we get

$$\int_0^T \delta(t - t')\, \mathrm{d}t = \sum_{i=1}^{N} f_i \Delta t = \frac{\Delta t}{\Delta t} = 1\,, \qquad \text{where } f_i = \begin{cases} \frac{1}{\Delta t} & \text{if } i = \left\lfloor \frac{t'}{\Delta t} \right\rfloor, \\ 0 & \text{otherwise}, \end{cases}$$

where f_i are the discrete samples of the function.

2.2   Random Signals

We mentioned above that we would like to randomly generate an input function $x(t)$. But what exactly do we mean by random? And how do we turn such a function into discrete time steps? A somewhat naïve attempt would be to select a timestep Δt, a maximum time T, and to just sample $N = \lfloor T/\Delta t \rfloor$ values from a Gaussian distribution. This kind of noise, where the individual points in time are independently sampled from a Gaussian distribution is called *white noise*.

Figure 1a shows what such a signal would look like for a time step $\Delta t = 1$ ms and $T = 1$.

Interestingly, the signal looks – qualitatively – the same in both the time and the frequency domain. Instead of generating this kind of signal in the time domain we could also create it in the frequency domain by generating random complex numbers for each Fourier coefficient

$$x(t) \sim \mathcal{N}(0, 1) \Leftrightarrow X(\omega) = X(-\omega) = a + ib, \text{ where } a \sim \mathcal{N}(0, 1) \text{ and } b \sim \mathcal{N}(0, 1),$$

where $X = \mathcal{F}[x]$ is Fourier-transformed signal. In particular, $X(\omega)$ is a function over the angular frequency $\omega = 2\pi f$.

It is impossible to generate true white noise (i.e., as $\Delta t \to 0$) in a physical system, since even finite portions of the signal contain an infinite amount of energy (remember that signal must change instantaneously between individual "quanta" of time). But even less fundamentally, we find that signals in the physical world are usually quite *smooth* – there are no sudden changes between values.

Hence, to model physical stimuli we may want to use *bandlimited white noise* – that is, we have some upper frequency $\hat{f}$ above which all frequency coefficients are zero. We can easily generate such a signal $x(t)$ by applying our above observation, namely that white noise in the time domain looking the same as in the frequency domain: we simply sample a set of complex numbers $z_1, \ldots, z_d$ corresponding (indirectly, see note) to the phases and powers of frequencies up to $\hat{f}$. We then apply the inverse Discrete Fourier Transformation (DFT) and voilà, we get our band-limited white noise function in the time domain $x(t)$ (fig. 1b). For a d-dimensional $\mathbf{x}(t)$ we simply repeat this process d times.

> **Note:** *Fourier Transformation.* The Fourier transformation decomposes a signal $x(t)$ into a sum of phase-shifted sine waves of different frequencies. Fourier's insight was that any (complex-valued) function $x(t)$ can be decomposed into an infinite sum of sine and cosine waves. It holds
>
> $$X(\omega) = \mathcal{F}[x](\omega) = \int_{-\infty}^{\infty} x(t)e^{-it\omega} \, dt = \int_{-\infty}^{\infty} x(t)[\cos(-t\omega) - i\sin(t\omega)] \, dt.$$
>
> Intuitively, for a fixed angular frequency ω, this function can be interpreted as computing the "similarity" (in terms of an inner product over functions) between the function $x(t)$ and the periodic signals $\cos(-t\omega)$ and $\sin(t\omega)$. Consider a specific value $X(\omega) = z = a + ib$, where a is the "real" value of the complex number z and b is the "imaginary" part of z. The similarity with the cosine is stored in a, the similarity with the sine is stored in b.
>
> The *power*, or "magnitude", of a particular frequency ω is given as $|z| = \sqrt{a^2 + b^2}$. The complex coefficients a and b also encode the *phase* ϕ of that particular frequency, where $\phi = \arctan2(b, a)$.
>
> *Inverse Fourier Transformation.* The Fourier transformation can be inverted by applying almost the same transformation again (note the flipped sign in the exponential):
>
> $$x(t) = \mathcal{F}^{-1}[X](t) = \int_{-\infty}^{\infty} X(\omega)e^{it\omega} \, d\omega = \int_{-\infty}^{\infty} X(\omega)[\cos(t\omega) + i\sin(t\omega)] \, d\omega.$$

Note: *Properties of the Fourier Transformation.*

- The Fourier Transformation is *linear*. In other words, it holds

$$\mathcal{F}[af + bg] = a\mathcal{F}[f] + b\mathcal{F}[g],$$

where a, b are scalars and f, g are arbitrary functions over time.

- It holds $X(\omega) = X(-\omega)$ exactly if $x(t)$ is a purely real valued signal. In other words, to ensure that the inverse Fourier Transformation outputs a real-valued signal we must choose $X(\omega)$ to be symmetric.

Discrete Fourier Transformation. We can write the above integral as a sum for discrete signals $x_0, \ldots, x_{N-1}$. We get

$$X_i = \sum_{k=0}^{N-1} x_k e^{-i \frac{i2\pi}{N} kn},$$

the so called Discrete Fourier Transform (DFT). You can compute the DFT of a signal $\mathbf{x} = (x_0, \ldots, x_{N-1})$ in Python using the following code (see this ▭ *this Jupyter notebook*).

```python
import numpy as np
dt = 1e-3                       # One millisecond time step
T  = 10.0                       # Maximum time: 10 seconds
ts = np.arange(0, T, dt)        # Sample points
N  = ts.size                    # Number of samples


# Generate a one Hertz sine wave
f  = lambda t: np.sin(2 * np.pi * t)
xs = f(ts)                      # Sample some function over time


# Compute the Discrete Fourier Transform using a fast algorithm
# called the "Fast Fourier Transformation" (FFT)
Xs = np.fft.fftshift(np.fft.fft(xs))


# Compute the frequency corresponding to each coefficient
fs = np.fft.fftshift(np.fft.fftfreq(N, dt))


# Reconstruct the original signal, throw away residual imaginary coefficients
xs_hat = np.real(np.fft.ifft(np.fft.ifftshift(Xs)))
```

The function `fftshift` arranges the Fourier coefficients such that the k-th coefficient X_k corresponds to the frequency $\frac{2k-N}{2T} = \frac{2k-N}{2\Delta tN}$ (for even N). `ifftshift` is the inverse of `fftshift`.

Aside: Have a look at the following resources if you would like to learn more about the Fourier transformation:

- Two videos by Grant Sanderson (3Blue1Brown) on the Fourier transformation:

    - ▣ But what is the Fourier Transform? A visual introduction.

    - ▣ But what is a Fourier series? From heat flow to circle drawings.

- An Interactive Introduction to Fourier Transforms by Jez Swanson:
  `http://www.jezzamon.com/fourier/index.html`

- Primer on the Discrete Fourier Transformation in the context of signal processing:
  `https://jackschaedler.github.io/circles-sines-signals/part3.html`

2.3 Computing Decoders

Given a discretised version of our input function $\mathbf{x}(t)$ and the recorded population spike trains $\mathbf{a}(t)$, we can finally construct matrices $\mathbf{X}$ and $\mathbf{A}$ and use these to compute a set of decoders $\mathbf{D}$ according to the least-squares solution we discussed in the last lecture

$$\mathbf{D}^\mathsf{T} = (\mathbf{A}\mathbf{A}^\mathsf{T})^{-1}\mathbf{A}^\mathsf{T}\mathbf{X}.$$

Note that it is not necessary to use the regularised version of this equation. As we discussed last lecture, regularisation is not necessary in case we have a large number of noisy samples – which is exactly what we have in this case. For example, if we record neural activities for a few neurons over $T = 10\,\mathrm{s}$ at a time step of $\Delta t = 1\,\mathrm{ms}$ we already have $10\,000$ samples for each neuron.

Note: *Interpretation of the decoder computation equation.* If we were to analyse the above equation more thoroughly, we see that we are implicitly approximating the average firing rate for each neuron, given a certain input $\mathbf{x}$.

In a sense, we are computing the rate approximation $G[J]$ of the individual neurons "on the fly". This has the consequence that our decoders $\mathbf{D}$ computed using the rate approximation $G[J]$ should still work in a spiking context. However, one advantage of this way of computing decoders is that it also works for more complicated neuron models for which we cannot easily compute a rate approximation $G[J]$.

Example: Decoding from two neurons Figure 2 shows an example of what we have discussed above. In this particular case, we choose neural parameters e_i, α_i, J_i^{bias} such that we have one neuron approximately covering the positive space of represented values, and another neuron covering the negative space of represented values. The decoder $\mathbf{d}$ for these two neurons is approximately

$$\mathbf{d} = \begin{pmatrix} 0.5\Delta t, & -0.5\Delta t \end{pmatrix}.$$

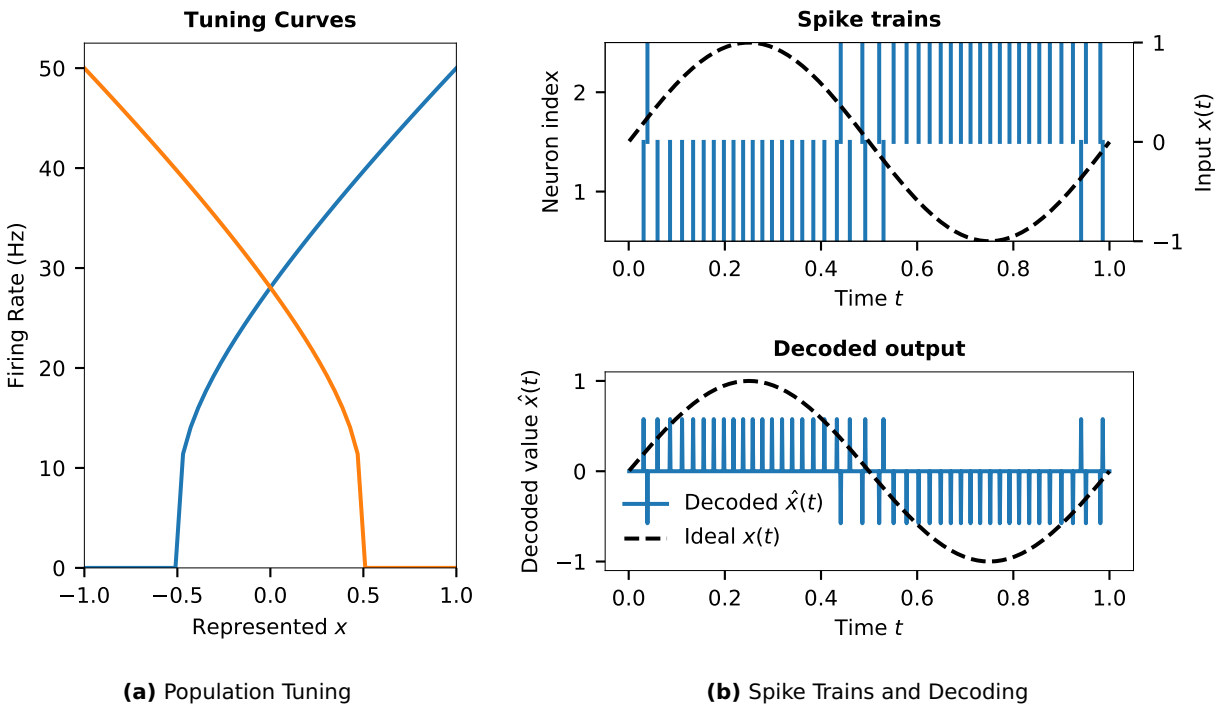**(a)** Population Tuning   **(b)** Spike Trains and Decoding

Figure 2: (a) Neural population tuning curves for two neuron. **(b)** Recorded spike trains for a sine-wave input (*top*) and decoded output (*bottom*). Time step of $\Delta t = 1$ ms. ⌨ *Code*
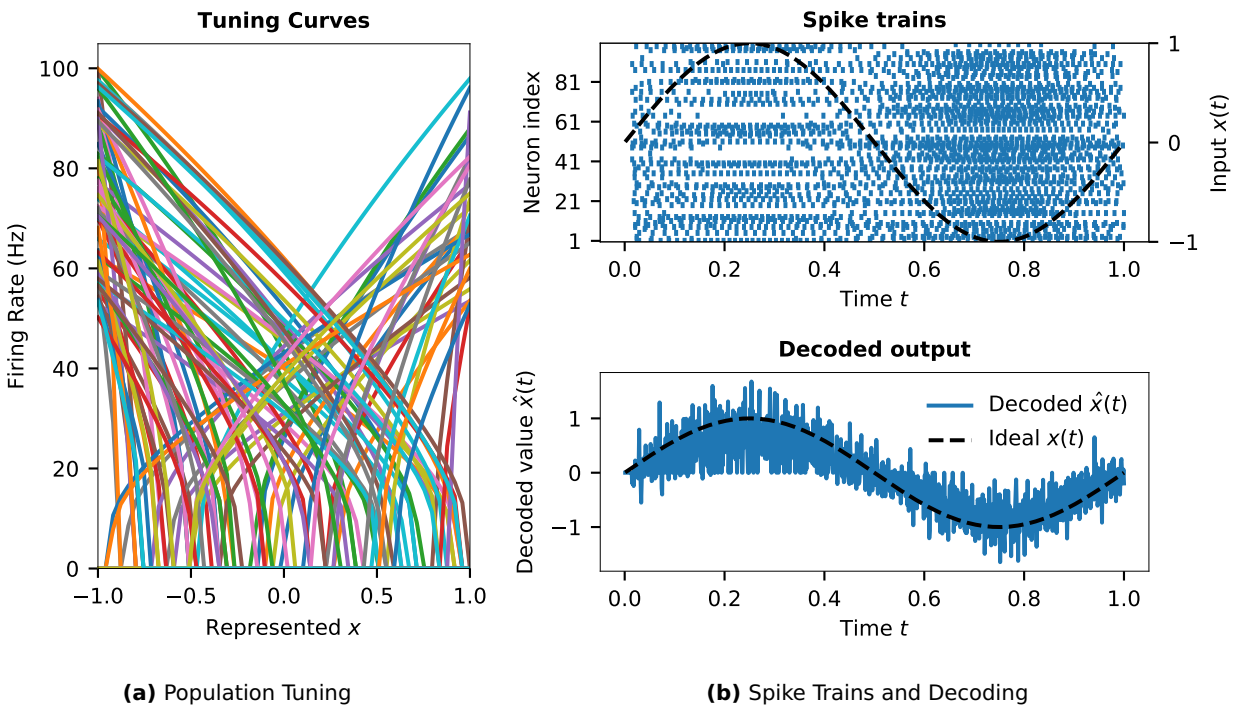

(a) Population Tuning   **(b)** Spike Trains and Decoding

Figure 3: (a) Neural population tuning curves for one hundred neurons. **(b)** Recorded spike trains for a sine-wave input (*top*) and decoded output (*bottom*). Time step of $\Delta t = 1$ ms. ⌨ *Code*
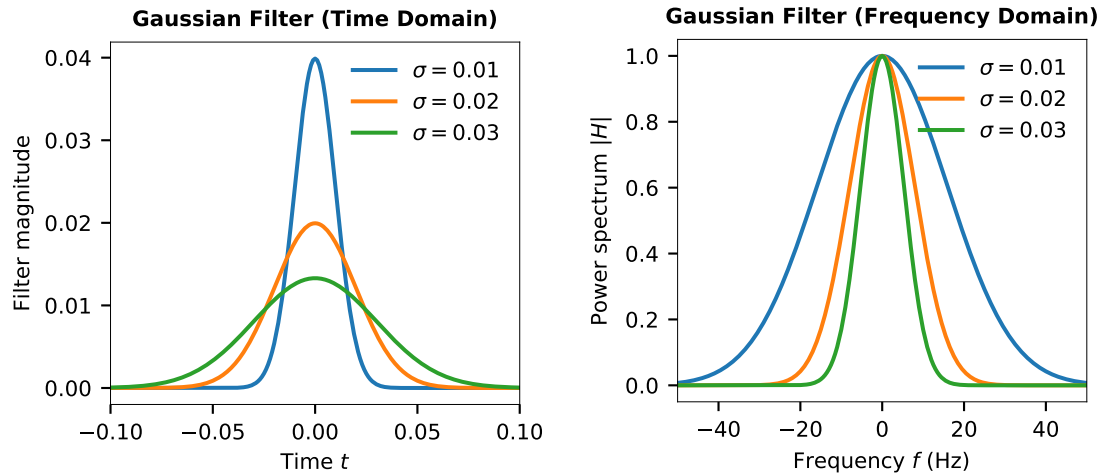
Figure 4: Gaussian low-pass filters in the time- and frequency domain. Note that the power spectrum $|H(\omega)|$ of a Gaussian filter is also a Gaussian. However, narrower (smaller σ) Gaussians in the time domain map onto broader (larger σ) Gaussians in the frequency domain.

Whenever we receive a spike from the first neuron (each spike modelled as a discrete Dirac-pulse of height $1/\Delta t$) we decode out 0.5, whenever we receive a spike from the second neuron we decode out -0.5. We do not have any information "in between spikes".

Example: Decoding from one hundred neurons If we increase the number of neurons, the chance of multiple neurons spiking in the same timestep increases. In this case, our decoded output actually starts to look more like what we would expect (fig. 3) – in particular, since there is a smaller change of being "in between spikes", the decoded output does not constantly return to zero. Unfortunately, this is solely an artefact of discretisation. As we decrease our timestep Δt, it gets less and less likely for two neurons to be spiking in exactly the same timestep.

3 Temporal Filtering

Our method of computing decoders seems to work, however, we have the general problem that we have no data whenever there is no spike, which is particularly problematic if we let $\Delta t \to 0$. As we decrease the time step, it will be less likely that multiple δ-functions coincide. One solution to this problem is to apply a temporal filter to our decoded signal that fills in values between individual spikes, or, if we regard the signal as a "noisy" version of the original signal, eliminates the noise.

3.1 Gaussian Filters

A commonly used type of filter is the Gaussian low-pass filter (fig. 4). This filter is defined in the time-domain by the equation

(Gaussian Low-Pass)

$$h(t) = c^{-1} \exp\left(\frac{-t^2}{\sigma^2}\right), \quad \text{where } c = \int_{-\infty}^{\infty} \exp\left(\frac{-t^2}{\sigma^2}\right) \mathrm{d}t. \quad (1)$$

The normalisation factor c ensures that the overall integral over h is one. This guarantees that the filter preserves the energy of a constant ("DC") input signal.

We apply a filter to a signal using the convolution operator "$*$". This operator takes two functions $f(t)$, $g(t)$ as an input and outputs a new function $(f * g)(t)$, where

$$(f * g)(t) \stackrel{\mathrm{def}}{=} \int_{-\infty}^{\infty} f(t - \tau)g(\tau)\,\mathrm{d}\tau.$$

Note: *Convolution in the Fourier Domain.* It holds $\mathcal{F}[f * g] = \mathcal{F}[f]\mathcal{F}[g]$. Put differently, convolution in the time domain is simple multiplication in the frequency domain. Take another look at the frequency-space representation of the Gaussian filter in fig. 4.

Given this equation, it should be fairly obvious why this kind of filter is called a "low-pass filter" – larger frequencies in the frequency representation of h are approximately zero, multiplication with the corresponding frequency coefficients of the original signal will effectively eliminate higher frequencies from the signal. Hence the filter only passes lower frequencies on.

In order to use the filter for our purposes we can simply filter the individual spike trains and then multiply with the decoders

$$\hat{\mathbf{x}}(t) = \mathbf{D}(\boldsymbol{\alpha} * h)(t).$$

Due to linearity of the convolution operation this is equivalent to first decoding the spike trains as shown in the previous sections and then applying the filter – which is computationally much more efficient since we only have to convolve once

(Temporal Decoding Equation)

$$\hat{\mathbf{x}}(t) = \big((\mathbf{D}\boldsymbol{\alpha}) * h\big)(t). \quad (2)$$

For a scalar signal $x(t)$ we have

$$\hat{x}(t) = \left(\left(\sum_{i=1}^{n} d_i a_i\right) * h\right)(t),$$

where d_i are the decoding coefficients and $a_i(t)$ are the individual neural activities.

As depicted in fig. 5a, this technique may work quite well as long as we have control over the input signal, and we scale the output correctly. However, once we switch to a band-limited white noise input signal (fig. 5b) the error increases significantly.
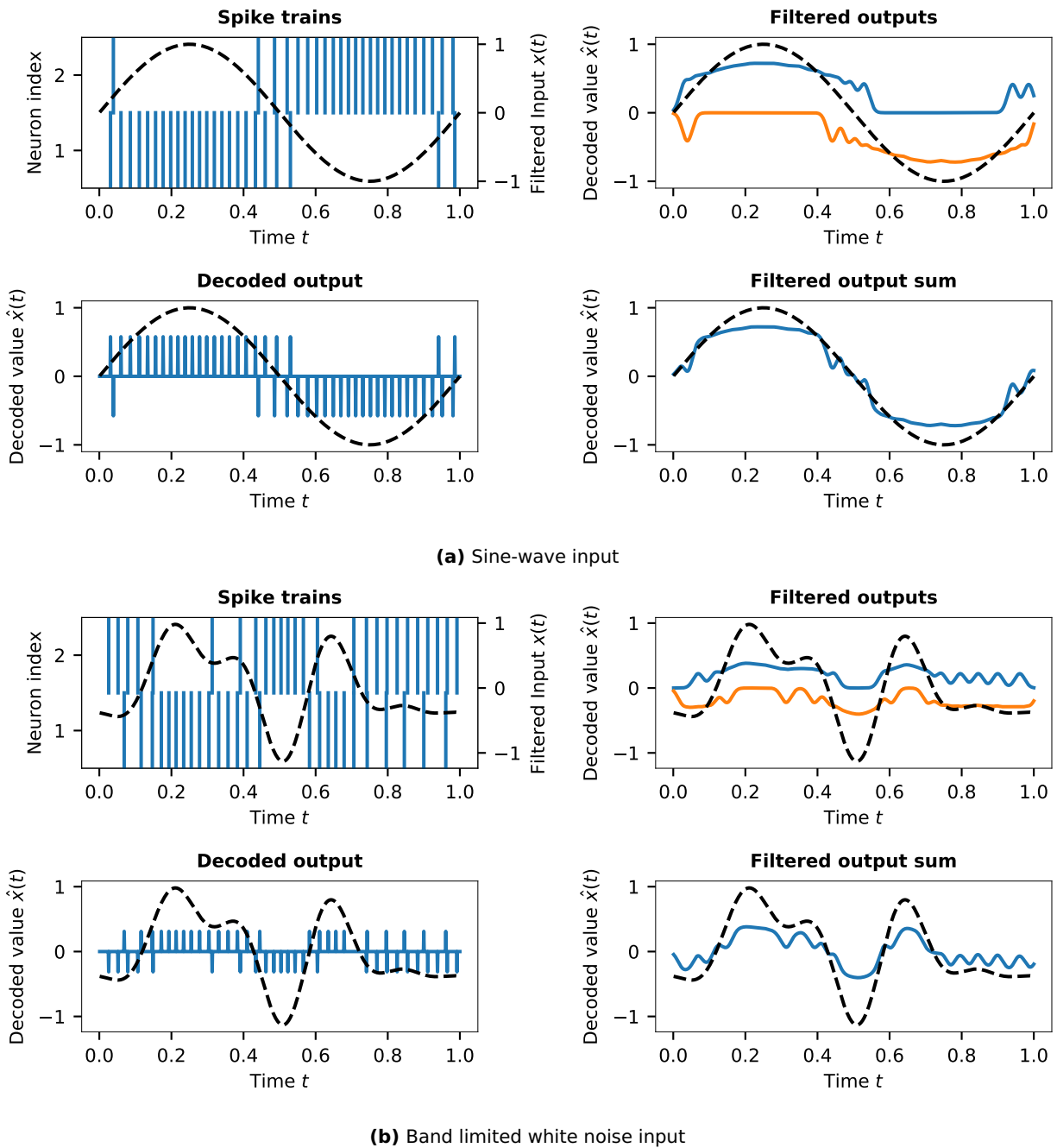
(a) Sine-wave input

(b) Band limited white noise input

Figure 5: Visualisation of spike train filtering using a Gaussian filter. **(a,b)** *Top left:* Spike train and input signal $x(t)$. *Bottom left:* Spike trains multiplied by the corresponding decoding weights. *Top right:* Individual filtered spike trains multiplied with the decoding coefficient for each neuron. *Bottom right:* filtered, decoded sum of the spike trains. ⌨ *Code*

3.2   Optimal filter

Can we do better than just using a standard Gaussian filter? Let's try to compute h more systematically, for example by phrasing the filter h we would like to use as the result of an optimization problem.

Special case: symmetric neuron tuning curves   In order to make the mathematical derivation a little easier, let us consider the special case of a population of neurons with only two neurons, in particular the two symmetric neuron tuning curves shown in fig. 2. We generated these tuning curves by selecting exactly the same α and J^{bias} parameters, but choosing the encoder e as 1 for the first neuron and -1 for the second neuron.

Since the neurons are symmetric opposites, we can assume that $d_1 = -d_2$, i.e., the decoding coefficient for the second neuron is exactly the negative of the first decoding coefficient. Hence, according to eq. (2), we have

$$\hat{x}(t) = \big((a_1 d_1 + a_2 d_2) * h\big)(t) = d_1\big((a_1 - a_2) * h\big)(t).$$

Folding the scaling factor d_1 into h we get

$$\hat{x}(t) = \big((a_1 - a_2) * h\big)(t) = (r * h)(t),$$

where $r(t) = a_1(t) - a_2(t)$ (r for "response"). Essentially, we treat the spikes from our two neurons as being signed – spikes from the first neuron have a positive sign, spikes from the second neuron have a negative sign.

> **Note:** In general, we could just use $r(t) = \langle \mathbf{d}, \mathbf{a}(t) \rangle$, although we cannot guarantee that h is the optimal filter in this case (since information about the spike source is lost). For populations representing more than one dimension we can just treat the individual dimensions separately.

Optimization problem   Once again, our goal is to minimize the decoding error $\hat{x}(t) - x(t)$. Correspondingly, we can start with the following least-squares problem:

$$h = \arg\min_{h} E = \arg\min_{h} \int_{-\infty}^{\infty} \big(x(t) - (r * h)(t)\big)^2 \, \mathrm{d}t.$$

Solving this equation is quite hard, since h is inside an integral and a convolution operator. We can simplify this equation by switching to the Fourier domain in which convolution becomes multiplication. We have

$$\hat{X}(\omega) = R(\omega)H(\omega),$$

and hence we want to find the H that minimizes the error

$$H = \arg\min_{H} E = \arg\min_{H} \int_{-\infty}^{\infty} \big|X(\omega) - R(\omega)H(\omega)\big|^2 \, \mathrm{d}\omega.$$

(a) Optimal filter

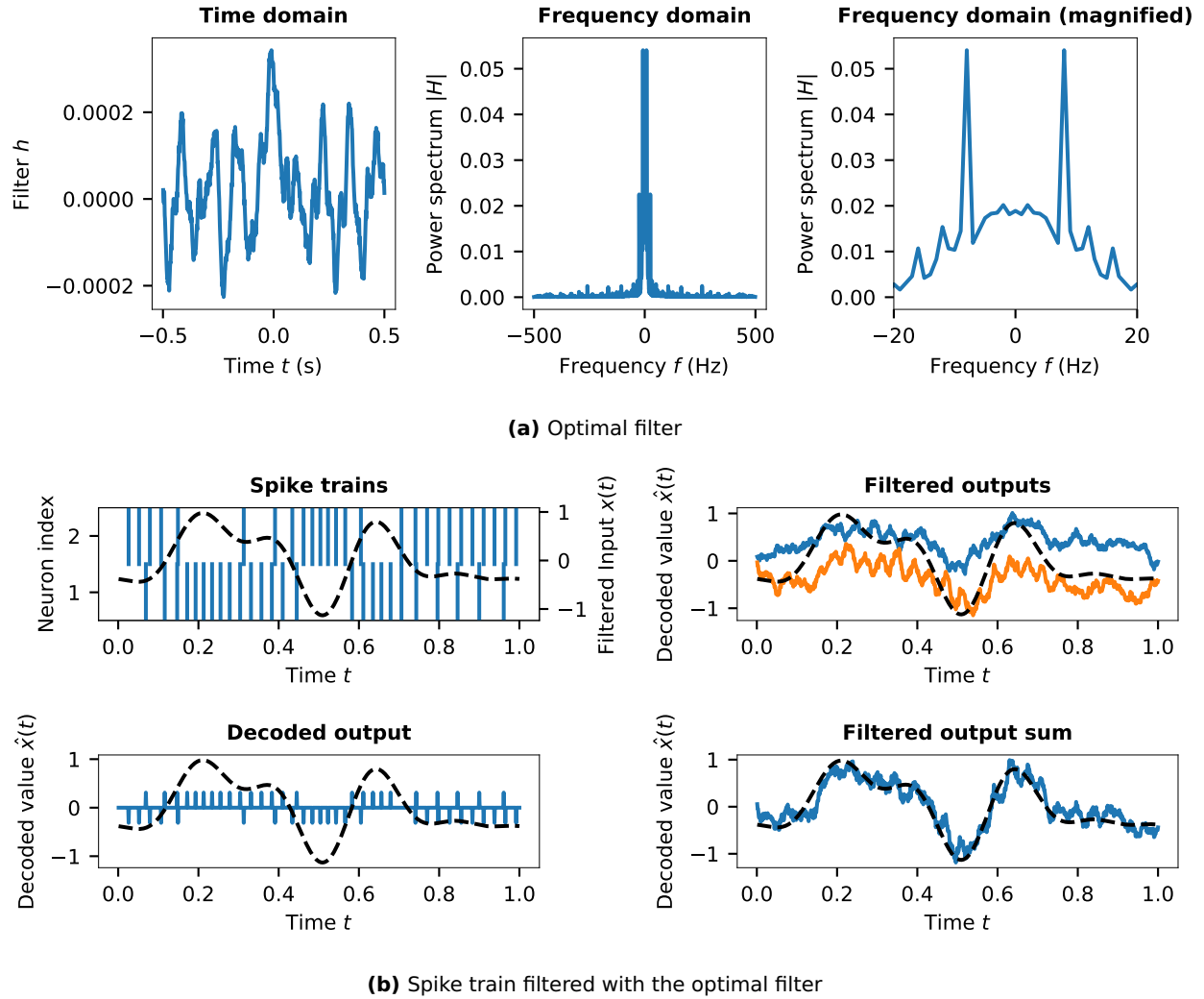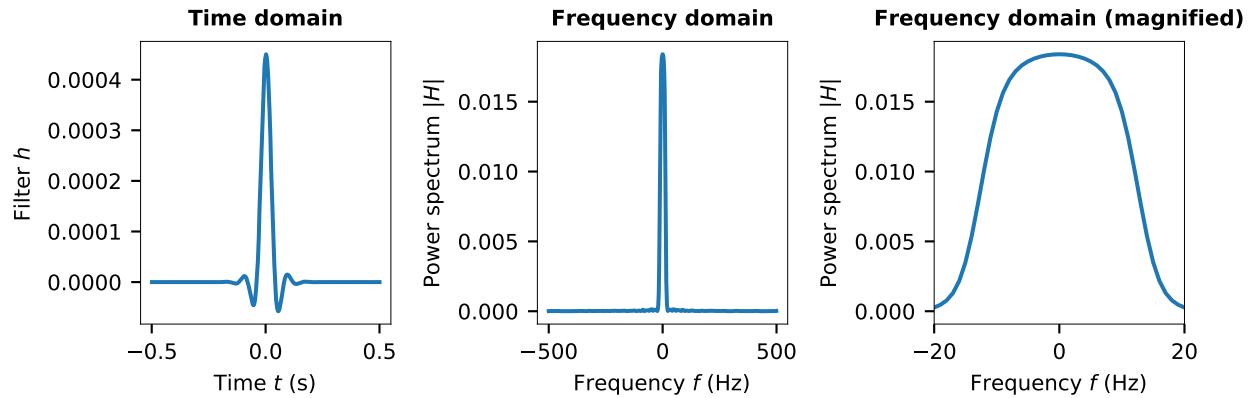**(b)** Spike train filtered with the optimal filter

Figure 6: Optimal filter (top) applied to the example from fig. 5b (bottom). ⌨ *Code*

Ignoring the fact that all the objects $R(\omega)$, $H(\omega)$, $X(\omega)$ are functions, this looks quite manageable! Since this equation is quadratic in $H(\omega)$ and the second derivative with respect to $H(\omega)$ is positive, we can find the optimal $H(\omega)$ by computing the derivative and equating the result with zero. Without going into the details (see [1], Chapter 4 and Appendix B.3 for more details), we get

$$H(\omega) = \frac{X(\omega)\overline{R}(\omega)}{|R(\omega)|^2}, \text{ where } \overline{z} = \overline{x + iy} = x - iy \text{ is the complex conjugate}.$$

We can apply this equation to the result of a discrete Fourier transformation of $x(t)$ and $r(t)$. In this case, we can treat $X(\omega)$ and $R(\omega)$ as vectors and all operations are element-wise.

Figure 6a shows the optimal filter for a 10 s bandlimited white noise input. This filter is then used on the input signal we already saw in fig. 5b. The result is shown in fig. 6b. Just judging visually, the error is reduced significantly. However, there is still quite some high-frequency noise present in the filtered output that is not in the original input signal.

12

(a) Improved optimal filter
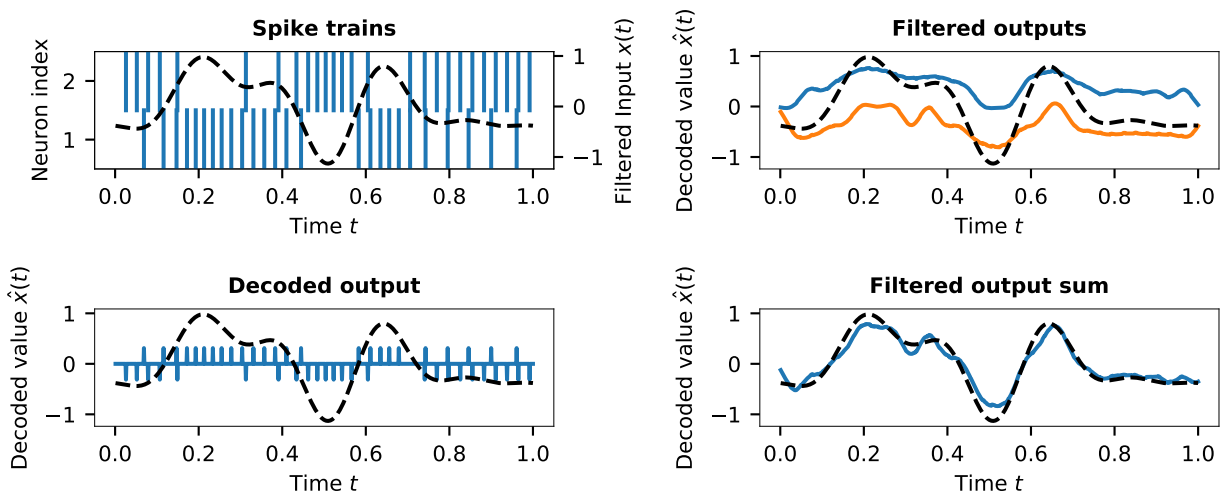
(b) Spike train filtered with the improved optimal filter

Figure 7: Optimal filter (top) applied to the example from fig. 5b (bottom). ⌨ *Code*

Improving the optimal filter   Looking at the optimal filter in the frequency domain (fig. 6a), we can see the filter itself appears to be "noisy". Could we eliminate this noise by filtering the optimal filter in the frequency domain? Doing this results in the following equation

(Optimal windowed filter)

$$H(\omega) = \frac{(X(\omega)R^*(\omega)) * W(\omega)}{|R(\omega)|^2 * W(\omega)} \,, \tag{3}$$

where $W(\omega)$ is a Gaussian filter (see above), also called a *window function*. As demonstrated in fig. 7, this filter successfully eliminates the noise.

Note: There is actually a nice mathematical derivation for eq. (3). Have a look at [1] and Appendix B.3 for more details.
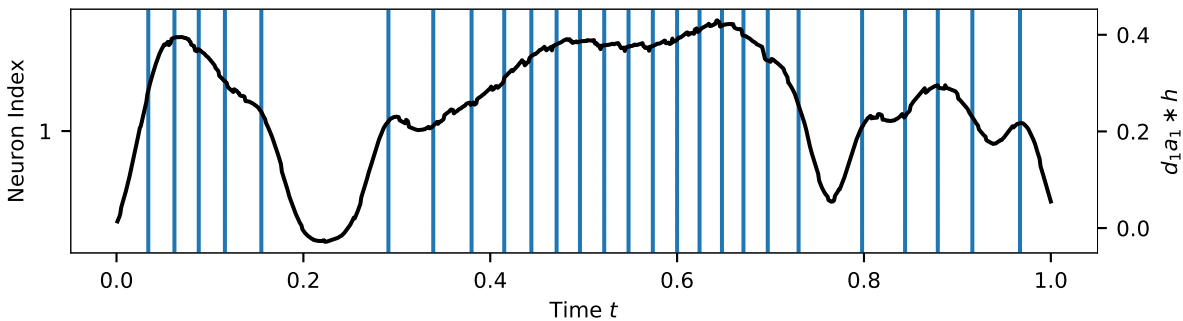
Figure 8: Optimal filter applied to the output of a single neuron. Note how future spikes influence the shape of the filtered activities.

4   Biological Filters

Let's have a closer look at the time-domain representation of the optimal filter depicted in fig. 7a. If we think of applying this filter in terms of convolution, then this filter weighs neural activity from the past (counter intuitively, the half of the plot for $t > 0$) and neural activity from the future (the half of the plot for $t < 0$) and combines them into an estimate of the value represented by our neural population (fig. 8). A filter that requires information about the future is called *acausal*. While this is fine for analysing recorded spiking activities after the fact, this kind of filter cannot be a model of what happens inside a neurobiological system – neurons are generally not able to look into the future.

Does this mean that we have to go back to the drawing board? Not quite. Our analysis shows that the optimal filter is essentially a low-pass filter. Hence, what we are looking for is some mechanism within a neurobiological systems that has a "smoothing" effect on the spike train. It turns out, that synapses provide this kind of filtering.

4.1   Synaptic Filters

Section 4.1 once more shows a schematic drawing of a synapse. Whenever a spike arrives at the pre-synapse, synaptic vesicles fuse with the cell membrane and release neurotransmitter into the synaptic cleft. The neurotransmitter diffuses to the post-synapses, where it triggers a chemical cascade that leads to the formation of a *post-synaptic potential*. Depending on the type of neurotransmitter released by the pre-synapse, this potential either has a positive, or a negative effect on the post-neuron's membrane potential. Positive potentials are called Excitatory Post-Synaptic Potential (EPSP), negative currents are called inhibitory post-synaptic potentials. The *currents* causing these changes in potential are called EPSC and IPSC, respectively. Section 4.1 shows such a recording of such potential in a motor neuron. As clearly visible, there is a short delay between the arrival of the pre-synaptic spike (start of the time axis), followed by a rapid increase in potential and a slow decay. This slow decay naturally acts as a low-pass filter.
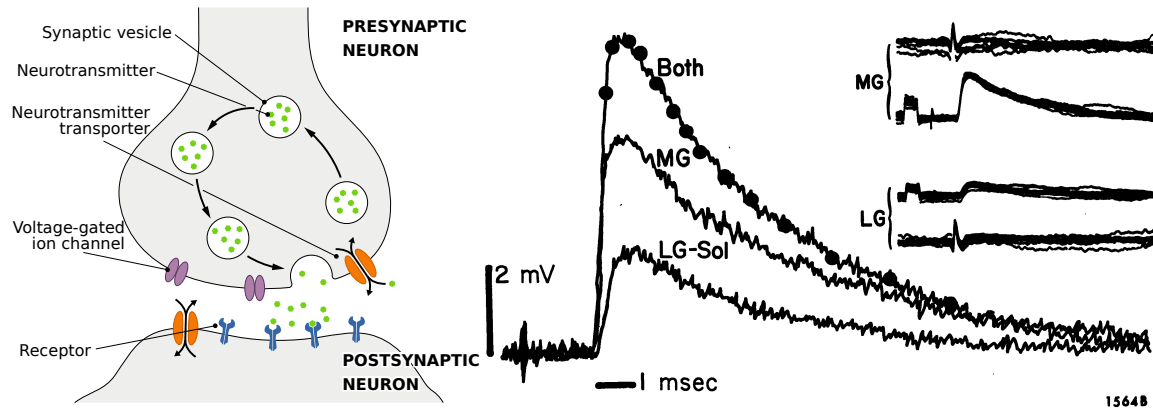
Figure 9: Synapses act as low-pass filters. **(a)** Textbook sketch of a single neuron. **(b)** Superposition of the motor potentials of two spikes arriving at a motor-neurons. Image copied from [2].

4.2 Models of Synaptic Filters

Post-synaptic potentials (and the corresponding currents) are commonly modelled as a n-th order exponential low-pass filter. In the time domain, this filter has the following equation

> *(Exponential Low-Pass)*
>
> $$h(t) = \begin{cases} c^{-1} t^n \exp^{-t/\tau} & \text{if } t \geq 0, \\ 0 & \text{otherwise}, \end{cases} \qquad \text{where } c = \int_0^\infty t^n \exp^{-t/\tau} \, dt. \qquad (4)$$

Figure 10 depicts this filter for different parameters of n and τ. Note how larger τ and n suppresses more high-frequency content. For $n = 0$ there is a discontinuous spike onset, larger n results in a smoother onset and delays the peak magnitude.

> **Note:** *Choice of parameters.* The parameters we use for n and – in particular – τ depend on neurophysiological data. Descriptions of individual brain regions will often include the synaptic time constant τ and or even traces of post-synaptic responses. While there is a vast variety of time constants, inhibitory GABA-A synapses tend to have a time constant of $\tau = 10$ ms, while excitatory NMDA synapses tend to have a time constant of $\tau = 5$ ms.

References

[1] Chris Eliasmith and Charles H. Anderson. *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems*. Cambridge, Massachusetts: MIT Press, 2003. 380 pp. ISBN: 978-0-262-55060-4.

[2] R E Burke. "Composite Nature of the Monosynaptic Excitatory Postsynaptic Potential." In: *Journal of Neurophysiology* 30.5 (1967), pp. 1114–1137. DOI: 10.1152/jn.1967.30.5.1114. eprint: https://doi.org/10.1152/jn.1967.30.5.1114. URL: https://doi.org/10.1152/jn.1967.30.5.1114.
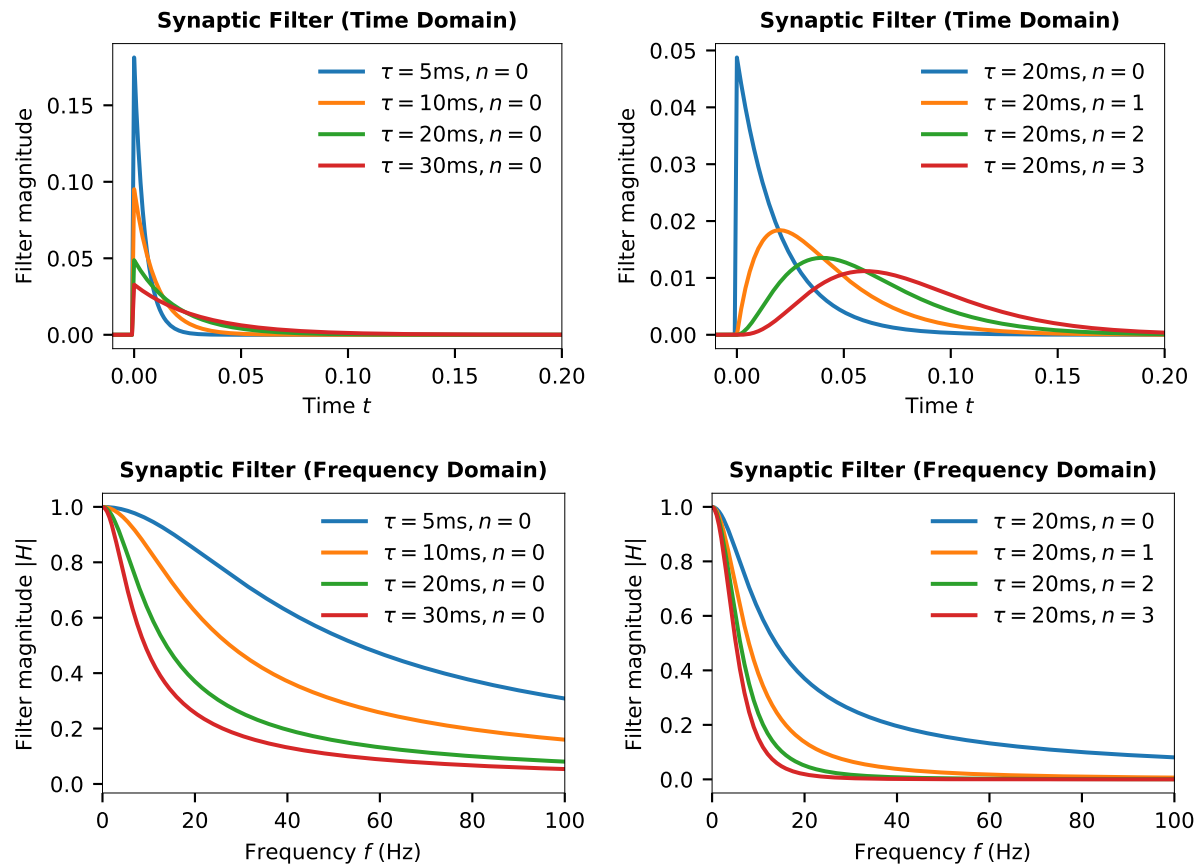
Figure 10: Exponential low-pass filters as a model of synaptic filtering. *Top:* Time-domain response for different parameters τ and n. *Bottom:* Frequency response of the corresponding filters. ⌨ *Code*