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

Accompanying Readings: Chapter 4 of Neural Engineering

Contents

1 Introduction **1**

2 Solving for a Spike Train Decoder **1**

2.1 Discretizing Spike Trains 2

2.2 Computing Decoders 3

3 Temporal Filtering **5**

1 Introduction

Note: We set out to build a theory of 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 this process, we assumed that each neuron in the population 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 take a stimulus over time and output a sequence of action potentials or *spikes*.

The output of a biologically plausible neuron model is also so 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 $t_i^1, t_i^2, \ldots$. 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 an integral 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 parts: 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 will gather a bunch of training data, consisting of a time-series of neural activity $\mathbf{\alpha}(t)$ 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 $\hat{\mathbf{x}}(t) = \mathbf{D}\mathbf{\alpha}(t)$.

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 $\{\mathbf{a}(\mathbf{x}_1), \ldots, \mathbf{a}(\mathbf{x}_N)\} \subset \left(\mathbb{R}^+\right)^n$, forming a matrix $\mathbf{A} \in \left(\mathbb{R}^+\right)^{n \times N}$.

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, bandlimited 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

$$J_i(t) = \alpha_i \langle \mathbf{e}_i, \mathbf{x}(t) \rangle + J_i^{\text{bias}},$$
$$a_i(t) = \sum_{k=1}^{\infty} \delta\left(t - t_i^k\right),$$

where t_i^k is the time of the i-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

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

with the appropriate super-threshold behaviour following spike production – whenever $v_i(t)$ reaches the threshold potential $v_{\text{th}} = 1$ we clamp the membrane potential to the reset potential $v_{\text{reset}} = 0$ for τ_{ref} seconds

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

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.

So, how do we assemble our matrices $\mathbf{X}$ and $\mathbf{A}$, which need to be constructed out of N discrete values? Well, 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 – each of these time slices can be interpreted as one of N samples for our matrix.

Note: *Discrete Dirac-δ.* While generating a discrete input function $\mathbf{x}(t)$ may be fairly straight forward (we can use the Discrete Fourier Transformation (DFT) to generate a bandlimited input function, see below), discretising the spike trains $\mathbf{a}(t)$ might be a little unintuitive, since each $a_i(t)$ 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. 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 discretise the integral. Assuming

we have a spike at time t' with $0 < t' < T$, we get

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

where f_i are the discrete samples of the function.

2.2 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 is it not necessary to use the regularised version of this equation. As we discussed last lecture, regularisation is not necessary if 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 analyse the above equation more thoroughly (🚧 *I will insert this here at a later point*), we see that we are implicitly approximating the average firing rate for each neuron, giving a certain input $\mathbf{x}$.

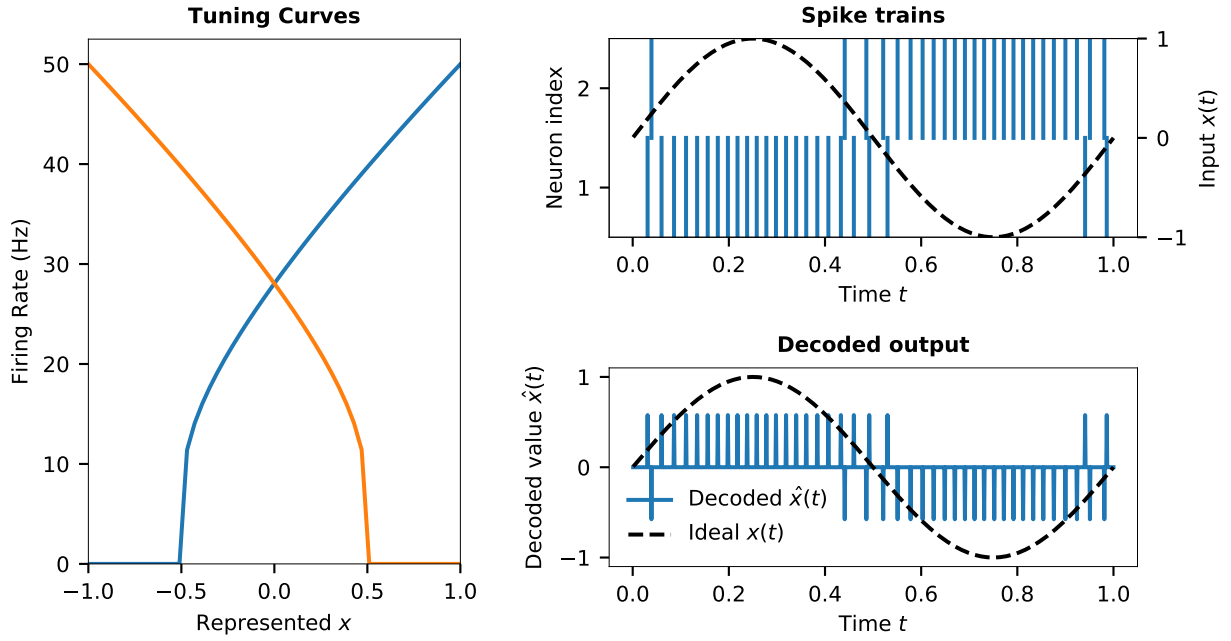
In a sense, we are computing the rate approximation $G[J]$ of the individual neurons "on the fly". This has the consequence that our decoders $\mathbf{D}$ computed using the rate approximation $G[J]$ should still work in a spiking context. However, one advantage of this way of computing decoders is that 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 1 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

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

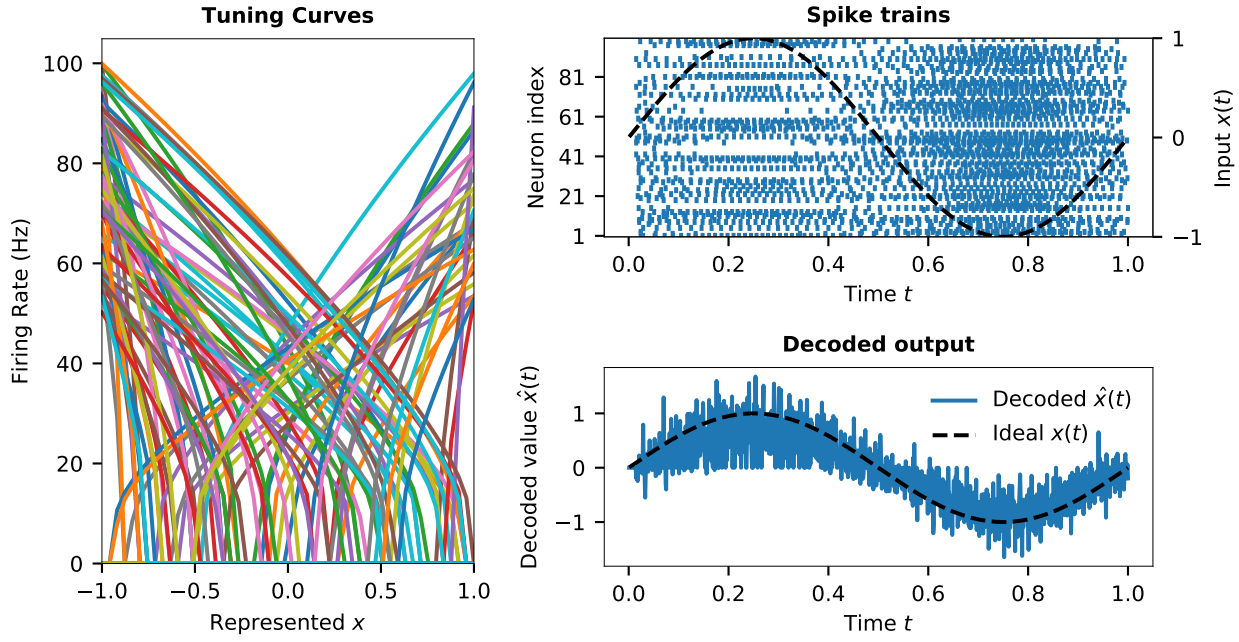
Whenever we receive a spike from the first neuron (each spike modelled as a discrete Dirac-pulse of height $1/\Delta t$) we decode out 0.5, whenever we receive a spike from the second neuron we decode out -0.5. We do not have any information "in between spikes".

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



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

Figure 1: (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 $\Delta t = 1\,\text{ms}$. *Code*



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

Figure 2: (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 $\Delta t = 1\,\text{ms}$. *Code*

decoded output actually starts to look more like what we would expect (fig. 2) – in particular, since 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

Filtering

- Problem: No data when there are no spikes
- Idea: Filter the output spikes with a filter h

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

- Scalar case:

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

- Convolution operator:

$$\big(f * g\big)(t) \overset{\text{def}}{=} \int_{-\infty}^{\infty} f(t-\tau) g(\tau)$$

- Which filter to chose? Gaussian filter (⇒ slides)

Random signals (See slides)

Optimal filter

- Special case: two neurons, same α and J^{bias} but different e (-1 and $+1$)
- Neurons are symmetric opposites, we can assume $d_1 = -d_2$
- Hence

$$\begin{aligned}\hat{x}(t) &= \big(a_1 d_1 + a_2 d_2) * h\big)(t) \\ &= d_1 \big((a_1 - a_2) * h\big)(t)\end{aligned}$$

- Fold the constant into h:

$$\begin{aligned}\hat{x}(t) &= \big((a_1 - a_2) * h\big)(t) \\ &= \big(\underbrace{r}_{\text{response}} * h\big)(t)\end{aligned}$$

- **Goal:** Find filter h that minimizes error:

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

- **Problem:** Convolution is annoying
- **Idea:** Fourier domain, convolution turns into multiplication

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

- Find H that minimizes error

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

- Derivative, set to zero:

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

  Optimal filter for a signal X and a response R. Optimize over a large range of values.

- Improvement: We know that we want to reject high frequencies. Add some windowing:

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

  where $W(\omega)$ is a Gaussian window function.