Lecture 5: Population-temporal representation and linear transformations

5.1 Putting temporal and population representation together

- Until now we have talked about temporal representation (spikes) and population (distributed) representation in neural systems as separate. Of course, they aren't in real systems.
- Given our definitions of these, constructing a unified 'population-temporal' code is very easy.
- Both population and temporal representation have been defined by a nonlinear encoding (neuron tuning curve) and linear decoding (optimal population decoders and PSCs).
- The encoding is as for the population code, but the nonlinearity, $G[\cdot]$, is a spiking nonlinearity:

$$\begin{aligned} a_i(\mathbf{x}(t)) &= G_i[J_i(\mathbf{x}(t))] \\ J_i(\mathbf{x}(t)) &= \alpha_i \left\langle \tilde{\boldsymbol{\phi}}_i \mathbf{x}(t) \right\rangle_m + J_i^{bias}. \end{aligned}$$

- Similarly, the form of the decoding will be just like that for temporal decoding:

$$\hat{\mathbf{x}}(t) = \sum_{i,n} \boldsymbol{\phi}_i h(t - t_{in}), \tag{5.1}$$

- The question is, what are the decoders when we have to think about the whole population rather than just a pair of neurons?
- The answer is: The same as in the regular population case. This is because we have used a LIF neuron to define $G_i[\cdot]$ in both cases. As a result,

$$\begin{aligned} a_i^{rate}(\mathbf{x}) &= \left\langle \sum_n h_i(t) * \delta(t - t_{in}) \right\rangle_T && (5.2) \\ &= \left\langle \sum_n h_i(t - t_{in}) \right\rangle_T && (5.3) \\ &= \left\langle a_i^{spiking}(\mathbf{x}) \right\rangle_T && (5.4) \end{aligned}$$

- So, we are using results from the steady state rate model (this is how we derived the response function, as you will recall) in conjunction with the non-steady state (spiking) model. This is how rate models are always used, since neural signals are dynamic.

- The most convincing demonstration that this is a good assumption is that we can build good models when we make the assumption.
- *One important note for implementation*: the $h(t)$ we find when determining optimal filters has to be normalized to have an area of 1. This is because both the temporal $h(t)$ and the population ϕ_i are used to account for the amplitude of the input. In physiological simulations, this means that the PSC is scaled to have an area of 1 before being multiplied by ϕ_i.
- Combining the population and temporal decoders gives:

$$\hat{\mathbf{x}}(t) = \sum_{i,n} \boldsymbol{\phi}_i(t - t_{in}) \tag{5.5}$$

- We can include noise in the population-temporal (PT) representation by introducing spike time jitter in the decoding process:

$$\hat{\mathbf{x}}(t) = \sum_{i,n} \boldsymbol{\phi}_i(t - t_{in} - \eta_{in}). \tag{5.6}$$

- The error we must minimize to find $\boldsymbol{\phi}$ is:

$$E = \left\langle \left[\mathbf{x}(t; \mathbf{A}) - \sum_{i,n} \boldsymbol{\phi}_i(t - t_{in} - \eta_{in}) \right]^2 \right\rangle_{\mathbf{A},\eta}. \tag{5.7}$$

- This can be solved either by determining the population and temporal decoders separately, or by finding $\boldsymbol{\phi}(t)$ directly using a Monte Carlo method.
- We use the separate approach because
  1. we can then use a non-optimal, biologically plausibly temporal decoder (the PSC)
  2. we can find the $\boldsymbol{\phi}$ analytically
  3. we can easily appy other (as yet unseen) analyses which help us understand population representation

5.2 Noise and precision

5.2.1 The effects of noise on the precision of temporal coding

- There are (at least) two ways to determine the effects of noise on optimal temporal filtering
  1. Pre-noising: add noise to a spike train, find the optimal filter, then decode the original (noiseless) spike train.

2. Post-noising: find the optimal filter using the (noiseless) spike train and then add noise to the spike train and decode it.

- As you might expect, the second approach results in worse performance. Even in this case, however, there is only a decrease of about 10% info transfer.
- This is expected because the noisy spike train is probably a lot like another random signal drawn from the same distribution and will thus give a similar optimal decoder. In contrast, post-noising compares a decoded noisy spike train to the original signal, so the noise should have an effect on the estimate.

5.2.2 Fluctuations as noise

- Conveniently, the use of spikes for neural coding results in fluctuations in the estimate of the represented variable that are analogous to noise. This is convenient because we've already analyzed the effects of noise on population representation. (Why have spikes if they just introduce noise? A: Communication over long distances)
- I won't go over the derivations in Appendix C.1 in detail, but let's look at some of the highlights of the derivation. Postsynaptic activity (i.e. the currents in the dendrites) is:

$$\alpha_i(x,t) = \sum_n h_i\left(t - n\Delta_i(x) - t_{i_0}\right), \tag{5.8}$$

where t_{i_0} is the time of the first spike. These t_{i_0} are evenly distributed random variables over $\Delta_i(x)$, where $\Delta_i(x)$ is the interspike interval (ISI) for neuron i given the value x. Note that even though the value of x is constant, the estimate of x will not be.

- The variance is:

$$\sigma^2_{\hat{x}(t)} = \left\langle \left[\hat{x}(t) - \langle \hat{x}(t) \rangle_T\right]^2 \right\rangle_T .$$

Which gives:

$$\sigma^2_{\hat{x}(t)} = \sum_i \phi_i^2 a_i(x) \left[\sum_m g_i\left(m\Delta_i(x)\right) - a_i(x) \right]. \tag{5.9}$$

where

$$g_i(\tau) = \int_{-\infty}^{\infty} h_i(t) h_i(t-\tau) dt.$$

- So, if the width of $h(t)$ is smaller than the ISI, there will be large fluctuations in the estimate of x because it's unlikely that different neurons have overlapping PSCs. As the width of the window increases, it's more likely there will be overlap

and that the resulting total current will be smoothed. When there is lots of overlap (i.e. $\Delta_i \sum g(m\Delta_i) \approx h * h$) then

$$\begin{aligned} \sum_m g(m\Delta_i) &\rightarrow a_i(x) \\ &= \frac{1}{\Delta_i} \end{aligned}$$

- The nice thing here, is that $\sigma^2_{\hat{x}(t)}$ is of the same form as the error due to noise, i.e., $\sigma^2 \sum_i \phi_i^2$. It may strike some as a bit odd that $\sigma_{\hat{x}}$ includes the ϕ_i but this is just a matter of how we did the derivation. In any case, we can write the error as:

$$\begin{aligned} E_{total} &= E_{static} + E_{noise} + E_{fluctuations} \\ &= \frac{1}{2}\left\langle \left[x - \sum_i a_i(x)\phi_i \right]^2 \right\rangle_x + \sigma_\eta^2 \sum_i \phi_i^2 + \sigma^2_{\hat{x}(t)}. \end{aligned}$$

or, better yet

$$\begin{aligned} E_{total} &= E_{static} + (E_{noise} + E_{fluctuations}) \\ &= \frac{1}{2}\left\langle \left[x - \sum_i a_i(x)\phi_i \right]^2 \right\rangle_x + \sigma^2 \sum_i \phi_i^2. \end{aligned}$$

where σ^2 accounts for both noise and spiking fluctuations.

- This means that the analyses we did on noise before are true for the spiking representation as well (i.e. error will decrease as $1/N$). You'll show this in the exercises.

5.3 Feedforward transformations

5.3.1 The communication channel

- Communication channel: the simplest kind of linear, feed-forward transformation: none at all.
- That is, sending some signal from one population to another population. (see slide)
- Begin constructing the model by writing down the representations in the two neural populations:

$$\begin{aligned} a_i(x) &= G_i\left[J_i(x)\right] && (5.10) \\ &= G_i\left[\alpha_i \tilde{\phi}_i x + J_i^{bias}\right] && (5.11) \\ \hat{x} &= \sum_i a_i(x)\phi_i^x, && (5.12) \end{aligned}$$

- There is a new notation with respect to the decoding weight, ϕ_i^x. I'll write the variable that the decoder is *for* (i.e., x) as a superscript on the decoder (i.e., ϕ_i), as this serves to disambiguate decoders. Disambiguation is important because characterizing transformations often requires identifying multiple decoders.

- For population b we have:

$$\begin{aligned} b_j(y) &= G_j\left[J_j(y)\right] && (5.13)\\ &= G_j\left[\alpha_j\tilde{\phi}_j y + J_j^{bias}\right] && (5.14)\\ \hat{y} &= \sum_j b_j(y)\phi_j^y. && (5.15)\end{aligned}$$

- To implement a communication channel using these two populations, we need to define the transformation: here, simply $y = x$. That is, we want our population y to represent whatever is represented by x.

- Now we can write the activities of b in terms of those of a by substituting x for y:

$$\begin{aligned} b_j(x) &= G_j\left[\alpha_j\tilde{\phi}_j x + J_j^{bias}\right] && (5.16)\\ &= G_j\left[\alpha_j\tilde{\phi}_j \sum_i a_i(x)\phi_i^x + J_j^{bias}\right] && (5.17)\\ &= G_j\left[\sum_i \omega_{ji} a_i(x) + J_j^{bias}\right], && (5.18)\end{aligned}$$

  where $\omega_{ji} = \alpha_j\tilde{\phi}_j\phi_i^x$. (note: We supposed $x = \hat{x}$).

- Voila, we have a connection weight matrix that performs the defined transformation.

- In neurobiological terms this eqn says that the instantaneous firing rate of a neuron in b_j is a result of the response to the current $J_j(x)$, which is determined by the sum of the dendritic currents (i.e., $\omega_{ij}a_i(x)$) plus some biasing background current J_j^{bias}. The dendritic currents (PSCs) are themselves determined by the product of their synaptic efficacy (ω_{ji}) and the firing of the presynaptic neuron $a_i(x)$.

- For the spiking model, we can write

$$\begin{aligned} \hat{x}(t) &= \sum_i a_i(x(t))\phi_i^x \\ &= \sum_{i,n} h_i(t - t_{in})\phi_i^x. \end{aligned}$$

- So, the spike trains of the b population are now determined as follows:

$$b_j(x(t)) = G_j\left[\alpha_j\tilde{\phi}_j x(t) + J_j^{bias}\right] \quad (5.19)$$

$$= G_j\left[\alpha_j\tilde{\phi}_j \sum_{i,n} h_i(t-t_{in})\phi_i^x + J_j^{bias}\right] \quad (5.20)$$

$$= G_j\left[\sum_{i,n} \omega_{ji} h_i(t-t_{in}) + J_j^{bias}\right], \quad (5.21)$$

  where $\omega_{ji} = \alpha_j\tilde{\phi}_j\phi_i^x$ as for the rate model.

- Of course, if we can do this computation, we can perform any scaling of x. We just include the scaling factor in the weight matrix. As well, we can/should include noise when finding the optimal decoders. If we don't bad things happen. The results of scaling by 1/2 (and ignoring noise) are as expected (see slide).

5.3.2 Adding scalar variables

- Let's again start by considering the scalar case. We assume our 3 populations are defined as before (see slide for connectivity). Now we perform a similar substitution:

$$\begin{aligned} c_k(x+y) &= G_k\left[\alpha_k\tilde{\phi}_k(x+y) + J_k^{bias}\right] \\ &= G_k\left[\alpha_k\tilde{\phi}_k\left(\sum_i a_i(x)\phi_i^x + \sum_j b_j(y)\phi_j^y\right) + J_k^{bias}\right] \\ &= G_k\left[\sum_i \omega_{ki}a_i(x) + \sum_j \omega_{kj}b_j(y) + J_k^{bias}\right], \end{aligned}$$

  where the weights $\omega_{ki} = \alpha_k\tilde{\phi}_k\phi_i^x$ and $\omega_{kj} = \alpha_k\tilde{\phi}_k\phi_j^y$ determine the two matrices needed for addition.

- So, given the firing rates $c_k(z)$ we find our estimate of z by using ϕ_k^z.

- The results of adding two periodic functions is as expected (see slide).

- Now, we have scalar multiplication and addition, which means we can find the weight matrices for any linear combination of scalars in a spiking network.

- Here's the recipe for doing linear transformations:

  1. Define the encoding and (representational) decoding rules for however many variables are involved in the operation.

2. Write the transformation to be performed in terms of these variables, one of which is the output variable (e.g., z above).
3. Write the transformation using the decoding expressions for all variables except the output variable.
4. Substitute this expression into the encoding expression of the output variable.

- Note: Rather doing this consecutively, it makes sense to implement various parts of a complex transformation in parallel.

5.3.3 Linear vector transformations

- Basically, there is nothing new here, but lets take a look. First, we define our representations to be of the form:

$$a_i(\mathbf{x}) = G_i\left[\alpha_i \left\langle \tilde{\boldsymbol{\phi}}_i \mathbf{x} \right\rangle_m + J_i^{bias}\right] \tag{5.22}$$

$$\hat{\mathbf{x}} = \sum_i a_i(\mathbf{x}) \boldsymbol{\phi}_i^{\mathbf{x}}. \tag{5.23}$$

- Note that the encoding (or 'preferred direction') vector, $\tilde{\boldsymbol{\phi}}_i$, isn't trivial as before. This essentially converts a vector of some physical magnitude into a scalar, via the dot product. In the scalar case, we did not need to explicitly denote this sort of conversion (though we did).

- Next, we define the (generic linear) transformation we want to implement in the network:

$$\mathbf{z} = C_1\mathbf{x} + C_2\mathbf{y}.$$

- Finally, we write the transformation using the representations and substitute that expression into the encoding rule for the output variable, $\mathbf{z}$:

$$\begin{aligned} c_k(C_1\mathbf{x} + C_2\mathbf{y}) &= G_k\left[\alpha_k \left\langle \tilde{\boldsymbol{\phi}}_k (C_1\mathbf{x} + C_2\mathbf{y}) \right\rangle_m + J_k^{bias}\right] \\ &= G_k\left[\alpha_k \left\langle \tilde{\boldsymbol{\phi}}_k \left(C_1 \sum_i a_i(\mathbf{x}) \boldsymbol{\phi}_i^{\mathbf{x}} + C_2 \sum_j b_j(\mathbf{y}) \boldsymbol{\phi}_j^{\mathbf{y}} \right) \right\rangle_m + J_k^{bias}\right] \\ &= G_k\left[\sum_i \omega_{ki} a_i(\mathbf{x}) + \sum_j \omega_{kj} b_j(\mathbf{y}) + J_k^{bias}\right], \end{aligned}$$

where $\omega_{ki} = \alpha_k C_1 \left\langle \tilde{\boldsymbol{\phi}}_k \boldsymbol{\phi}_i^{\mathbf{x}} \right\rangle_m$ and $\omega_{kj} = \alpha_k C_2 \left\langle \tilde{\boldsymbol{\phi}}_k \boldsymbol{\phi}_j^{\mathbf{y}} \right\rangle_m$.

- More generally, we can allow the scalars, C_1 and C_2, to be the matrices, $\mathbf{C}_1$ and $\mathbf{C}_2$. In this case, the resulting weights are of the form $\omega_{ki} = \alpha_k \left\langle \tilde{\boldsymbol{\phi}}_k \mathbf{C}_1 \boldsymbol{\phi}_i^{\mathbf{x}} \right\rangle_m$. Using matrices rather than scalars permits the incorporation of various kinds of linear operations such as rotation and scaling.
- As before, we can include spiking neurons in this transformation by allowing $a_i(\mathbf{x}) = \sum_n h(t - t_{in})$.
- This network does a good job of vector addition using the representations we have defined (see slide). There is a bit of a transient when using spiking neurons because of the dynamics of τ_{syn}.
- If we want to improve the network's performance, we can simply add neurons.
- There are indications that this kind of network (doing vector addition/subtraction) may be used in frontal eye fields for control of saccades.