SYDE 556/750
Simulating Neurobiological Systems
Lecture 5: Feed-Forward Transformation

Andreas Stöckel

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

January 30, 2020

Accompanying Readings: Chapter 6 of Neural Engineering

Contents

1 Introduction

Note: Until now, we have been concerned with *representation* in individual populations of neurons. However, we ultimately want to be able to build neural *networks*. This means that we have to find a systematic way of connecting neural populations. Optimally, the connections should be chosen in such a way that information represented in the populations is *transformed* in a useful way.

We postulated that groups of neurons represent d-dimensional quantities $\mathbf{x}$ using nonlinear encoding. In particular, the current J_i that is being injected into the i-th neuron of a population is defined as

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

For rate neurons, this input current is translated into a spike rate $a_i = G[J_i]$. In the context of spiking neurons, the input current is translated into a spike train $a_i(t)$, a sum of Dirac-δ pulses.

We then went on to postulate that the value that is represented by a population $\hat{\mathbf{x}}$ can be estimated using a linear decoder $\mathbf{D}$. For spiking neurons, we furthermore added a filtering step using a filter h

$$\hat{\mathbf{x}} = \mathbf{D}\boldsymbol{\alpha} \quad \text{for rate neurons,} \qquad \hat{\mathbf{x}}(t) = \big((\mathbf{D}\boldsymbol{\alpha}(t)) * h\big)(t) \quad \text{for spiking neurons.}$$

Note: *Review: Decoder computation methods.* We described two methods for the computation of the decoder matrix $\mathbf{D}$.

For the first method, we generated a random set of samples arranged in a matrix $\mathbf{X}$. Using the spike rate approximation $G[J]$, we computed an activity matrix $\mathbf{A}$. We then obtained $\mathbf{D}$ using the solution to the L_2-regularised least-squares problem

$$\mathbf{D}^T = (\mathbf{A}\mathbf{A}^T + \sigma^2 \mathbf{I})^{-1} \mathbf{A}\mathbf{X}^T,$$

where we estimate σ to match the amount of (Gaussian) noise that is present on top of the spike rate estimates.

The second solution was to use a random input function $\mathbf{x}(t)$, and the recorded population spike trains $\boldsymbol{\alpha}(t)$. We discretised these functions into matrices $\mathbf{A}$ and $\mathbf{X}$ and used the unregularised solution to the least-squares problem to obtain $\mathbf{D}$

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

As mentioned in the last lecture, these two methods should result in approximately the same decoders. While the second method is technically superior, as it accurately characterises the noise present in the neural population, the first method is computationally much cheaper. This is why – from now on – we will use the first method to compute the decoders. We will then use these decoding matrices in conjunction with spiking neurons.

Of course, just describing an individual population of neurons is not really useful when building large-scale brain models. We ultimately would like to describe how information is *transformed* while it is sent from one population of neurons to another. In the Neural Engineering Framework, connections between neural populations are theorised to perform these transformations.

NEF Principle 2 – Transformation
Connections between populations describe transformations of neural representations. These transformations are functions of the variables that are represented by neural populations.

When looking at transformations from the perspective of large-scale modelling, researchers are usually confronted with two very different questions.

- **How do brains *learn* transformations?** In other words, how are the connections weights between neuron populations formed in such a way that they implement a desired task.
- **What are the *optimal* set of connection weights that computes a certain transformation?** Here, we assume that a brain has already learned to optimally perform a certain task. In that case, we would just like to know what the corresponding connection weights that the system could have learned are. We would then like to use these weights in our model – essentially, we are building a model of a system that is already an expert at solving a certain task.

For now, we will mostly concern ourselves with the second question, i.e., we are trying to build models of "adult" or "expert" systems already capable of solving a certain task. We postulate what the transformation may be that the system has learned and compute the optimal weights that implement this transformation.

We will talk about *learning*, i.e., building a system that learns connection weights while it is being executed in a later lecture.

2 Building a Communication Channel

The simplest possible transformation is the identity function. Given to neural populations A and B, A representing $\mathbf{x}$, and B representing $\mathbf{y}$, we would like to send the value $\mathbf{x}$ from A to B, such that when simulating the network $\mathbf{x}$ and $\mathbf{y}$ are approximately the same. This type of transformation is also called a *communication channel*, since we are merely sending information from one point in the system to another point in the system without altering it.