Neuronal Representation

BCI Signal Decoding

Raihan Abdul Vaheed
Jakeb Chouinard

October 18, 2024

1 Introduction

It's Time for Neuron Math

It may not be evident how exactly a population of spiking neurons can come to represent values. On top of that, it might not make sense when we consider how math can be performed between populations of neurons. To address this, we'll introduce the basics of the Neural Engineering Framework. This should supply the basis by which we can develop a more comprehensive SNN model.

This is largely intended to be an internal document. If you would like to distribute or share this document with anyone outside of the BCI Signal Decoding project team and/or WAT.ai Admin, please ask the TPMs for approval first.

2 LIF Neurons

As you may recall from the *Neurons - BCI Signal Decoding* resources document, Leaky Integrate-and-Fire is a common model by which SNNs simulate neurons. When we consider making a population of LIF neurons, we want to ensure that each neuron we simulate is plausible and fits within the schema of the population. That is, we want to make sure that all of the neurons fall into the same range of maximum firing rates and that there is a proper distribution of coverage for provided stimulii. Moving forward, we'll consider input stimulii to be a normalized value between -1 and 1. This could be stimulus along the lines of a bar of light oriented at $0\,\deg$ compared to a bar of light oriented at $\frac{\pi}{2}$. Within a population of neurons responsible for extracting object orientation, some will fire for the flat orientation, and some will fire for the vertical orientation. Their firing rate will also increase as the presented stimulus becomes more aligned with the neurons' preferred stimulus. Figure 1 shows the tuning plot of a population of three neurons with varying neuron parameters: encoders, $\mathbf{e}$, gains, α, and biases, J_{bias}. A tuning plot is a tool by which we can observe the firing rate of a set of neurons when presented with different stimulus values.

For those that do not recall the LIF estimates from the Neuron resource document: gained, post-encoder current accounting for bias is:

$$J = \alpha\langle \mathbf{x}, \mathbf{e}\rangle + J_{\text{bias}}$$

Similarly given an absolute refractory time period – τ_{ref} – and the neural circuit time constant – τ_{RC}, an estimate for firing rate can be expressed as:

$$G[J] = \begin{cases} \frac{1}{\tau_{\text{ref}} - \tau_{\text{RC}} \log\left(1 - \frac{J_{th}}{J}\right)} & J > J_{th} \\ 0 & \text{else} \end{cases}$$

2.1 LIF Parameters

2.1.1 Encoders

For a 1-Dimensional input case, encoders for neurons are uniformly sampled from $\{-1, 1\}$. For a 2-Dimensional input case, encoders for neurons are uniformly sampled from:

$$\{[\sin(\theta), \cos(\theta)], \forall\theta \in [0, 2\pi)\}$$

More intuitively, a 2-Dimensional input encoder is uniformly sampled from the unit circle. You may see where this is going. For a 3-Dimensional input case, encoders for neurons are uniformly sampled from the unit sphere (an equation will not be provided because yeah). By extension, for the N-Dimensional input case, encoders for neurons are uniformly sampled from an N-dimensional hyper-sphere – that is, its magnitude will always be 1,

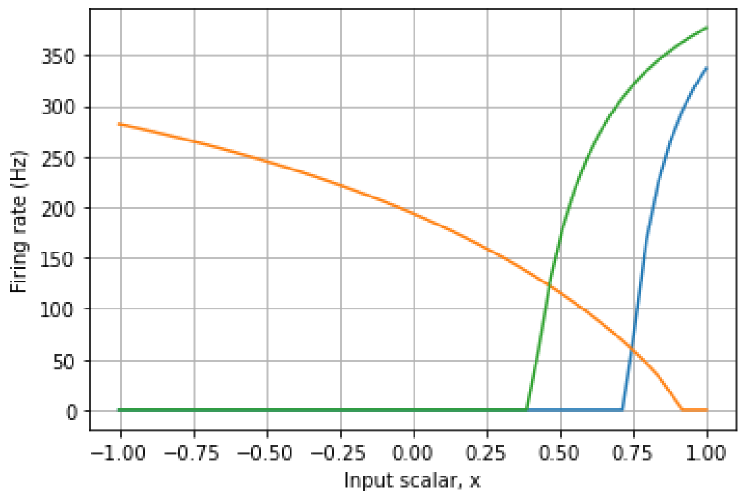

Figure 1: Three LIF Neurons with Varying Encoders and Biases

and it will have components in all N dimensions (even if those components are equal to 0). This lends to neurons in an N-dimensional input case being more or less associated with different input dimensions and whether or not the input value is positive or negative. In the case of Figure 1, the three neurons are only sensitive to the singular input dimension. The orange neuron exhibits a negative encoder value, as its firing rate increases with an increasingly negative input, whereas the green and blue neurons exhibit a positive encoder value, as their firing rates increase with an increasingly positive input.

2.1.2 Gains

Gains correlate to the rate at which a neuron's firing rate increases as the magnitude of the input increases. We can again reference Figure 1 for a more intuitive understanding. In order of increasing gain, the neurons are: orange, green, and blue.

2.1.3 Biases

Biases correlate to an offset of gained post-encoder values. Supposing some minimum current to cause neural spiking, J_{th}, J_{bias} is the difference between J_{th} and the minimum gained post-encoder value that causes spikes. That is, if the $\mathbf{x}$ values increase in the direction of the encoder $\mathbf{e}$, the spiking will become non-zero.

$$J_{\text{bias}} = J_{\text{th}} - \alpha\langle \mathbf{x}_0, \mathbf{e}\rangle$$

2.2 Population Construction

One thing that may not be entirely evident is that an improper combination of neuron parameters could result in a tuning plot for one neuron that results in a firing rate unreasonably high in comparison to the other neurons. For instance, a neuron with a positive encoder, a high gain, and a negative current bias would result in a tuning plot that starts lower than the lowest possible value of the input and increases rapidly until it eventually saturates at some maximum firing rate far above other neurons' values.

How can we avoid these infeasible results? A sampling of desired neuron properties, such as the maximum firing rate, a_{max}, and the input value offset, ξ, can result in correctly constrained tuning plots. Once again referring to Figure 1, we see a_{max} values between (approximately) 270 and 380 and ξ value of approximately 0.37 (green), 0.70 (blue), and 0.85 (orange). A particularly nasty derivation process can be followed to calculate α and J_{bias} from these parameters – the results of which are shown below:

$$\alpha = \frac{1}{1-\xi}\left(\left(1 - \exp\left(\frac{\tau_{\text{ref}} a_{\text{max}} - 1}{\tau_{\text{RC}} a_{\text{max}}}\right)\right)^{-1} - 1\right)$$

$$J_{\text{bias}} = 1 - \alpha\xi$$

This permits a construction of populations through uniform sampling of maximum firing rates and input value offsets as well as uniform sampling of encoder values. From these, we can reconstruct the gains and current biases for each neuron fairly trivially.

2.3 Population Connection

When neurons fire, their superimposed post-synaptic output becomes other neurons' inputs. Similar to traditional artificial networks, these values are linearly weighted – something called a decoder weight under the Neural Engineering Framework. If we define the output of an n^{th} layer's neuron $y_{n,i} \forall i \in \{1 \cdots N\}$, an arbitrary neuron's input in the next layer $\mathbf{x}_{n+1,j}$ is the weight-decoded combination of the previous layer's output where $d_{i,j}$ is the decoder weight between the n^{th} layer's i^{th} neuron and the $(n+1)^{th}$ layer's j^{th} neuron:

$$\mathbf{x}_{n+1,j} = \sum_{i=1}^{N} (d_{i,j} y_{n,1})$$

This results in the $(n+1)^{th}$ layer's j^{th} neuron experiencing an input of $\langle \mathbf{x}_{n+1,j}, \mathbf{e_{n+1,j}} \rangle + J_{\text{bias,n+1,j}}$ – so on and so forth.

3 The Neural Engineering Framework

The **N**eural **E**ngineering **F**ramework is a set of underlying modeling principles that are used to develop Spiking Neural Network models. A complete version of the NEF is presented in *Neural Engineering* by Chris Eliasmith and Charles H. Anderson. For further reference, a paraphrased version of the NEF is presented on the CNRGlab website here. All of the following information in this section is paraphrased and/or adapted from this source.

There are three central principles to the NEF [1]:

1. Neural representations are defined by the combination of nonlinear encoding and weighted linear decoding
2. Transformations of neural representations are functions of variables represented by neural populations determined using transformational decoding
3. Neural dynamics are characterized by considering neural representations as control theoretic state variables

3.1 P1: Encoding and Decoding

Referring to previous sections, a population's neurons encode their input through nonlinear spiking at a rate proportional to its input and internal parameters. The spike-train is convolved with a low-pass filter and overlapping spikes have superimposed outputs. The outputs are linearly weighted as inputs to the following neurons. Identity decoders – decoders such that the output of the population is equal to the input – can be calculated by analyzing each neuron's repsonses to a provided stimulus. Supposing a set of d-Dimensional N input values, $\mathbf{x}$, we can calculate the activities of a population of n neurons and formulate them as a matrix $\mathbf{A}$ where each column is a neuron's response to the N inputs values. Summarized variables are as follows:

$$\mathbf{x} \in \mathcal{R}^{(d,N)}, \quad \mathbf{A} \in \mathcal{R}^{(n,N)}$$

$$\mathbf{x} \in \mathcal{R}^{(d,N)}, \quad \mathbf{A} \in \mathcal{R}^{(n,N)}$$

We can calculate optimal identity encoders through the following Least-Squares Regression problem:

$$\mathbf{D}^T \approx \left(\mathbf{A}\mathbf{A}^T\right)^{-1} \mathbf{A}\mathbf{X}^T$$

Of course, we encounter some problems when the pseudo-inverse is non-invertible, lending us to add some diagonal noise term – this also aids in our ability to calculate decoders while some noise exists:

$$\mathbf{D}^T \approx \left(\mathbf{A}\mathbf{A}^T + N\sigma^2\mathbf{I}\right)^{-1}\mathbf{A}\mathbf{X}^T$$

3.2 P2: Functional Transformations

Different decoding weights can be calculated in such a way that the connections between populations can calculate nonlinear functions of the input populations' decoded values. Functional decoders can be calculated in a very similar way to identity decoders. Let $\ell : \mathcal{R}^{(d_1,N)} \to \mathcal{R}^{(d_2,N)}$:

$$\mathbf{Y} = \ell(\mathbf{X})$$
$$\mathbf{D}_\ell{}^T \approx \left(\mathbf{A}\mathbf{A}^T + N\sigma^2\mathbf{I}\right)^{-1}\mathbf{A}\mathbf{Y}^T$$

3.3 P3: State-Space Dynamics

This one's a little bit complicated, especially if one does not have a control theory background. For those that don't know, a general matrix formation of a state-space dynamics system is as follows:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$

Approximately equivalent neural dynamics matrices can be determined as follows:

$$\mathbf{A}' = \tau\mathbf{A} + \mathbf{I}$$
$$\mathbf{B}' = \tau\mathbf{B}$$

These can be inserted back into firing rate estimates:

$$G[J] = G[\alpha\langle\mathbf{e}(h(t) * [\mathbf{A}'\mathbf{x}(t) + \mathbf{B}'\mathbf{u}(t)]\rangle + J_{\text{bias}}]$$

Of particular note is that the state-space version of an input nicely encapsulates the system dynamics in a way that has been well-analyzed by control theory engineers for a long time. This enables easier overall analysis of the neurobiological simulation system.

3.4 Methods

While a highly iterative and difficult process, a generic methodology for the development of a model that simulates a neurobiological system is as follows:

1. System Description: Functional and neuroanatomical information should be used to create an overarching architectural, functional, and representational understanding of the system to be modeled. This includes interconnectivity and mathematical representations that describe the desired functionality. Our goal is to convert the neurobiological system into a system of mathematical representations.

2. Design Specification: Implementation constraints are important to consider in the model development. This involves understanding necessary representation kinds, precision, noise ratios, etc. This should all be made based on the basis of available data.

3. Model Implementation: Our goal here is to compute linear decodings that meet design specification and the determine connection weights between neurons. This is also a good step to determine the resolution at which we will simulate different parts of the model.

4 Summary

Pretty neat, right? We can compute functions over neural representations using things like tuning plots and observed activities in combination with simple regularization. Of course, we don't always *know* what kind of transformation we need. In the next resource paper, we'll discuss learning mechanisms and their application to SNNs.

All the best,

Jake and Rai

References

[1] T. Bekolay and A. Voelker, *Overview of the nef.* [Online]. Available: `https://compneuro.uwaterloo.ca/research/nef/overview-of-the-nef.html`.