SYDE 556/750
Simulating Neurobiological Systems
Lecture 12: Biological Details

Andreas Stöckel

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

March 31, 2020

Accompanying readings: Stöckel and Eliasmith, 2019 [1]

Contents

1 Introduction

Note: Within the Neural Engineering Framework, we are making some "biologically implausible" assumptions. In this lecture, we discus some of these assumptions and present ways to build networks that incorporate more biological detail.

Biology puts a few more constraints onto neural networks than what we assumed when deriving the equations underlying the NEF principles. The following list discusses three particular constraints we ignored. We will discuss how to extend the NEF to take these restrictions into account, resulting in potentially more biologically plausible networks.

- **Dale's principle: Neurons are either excitatory or inhibitory.** In biology, each type of neuron can only produce one type of neurotransmitter [2]. Most neurotransmitters have either an excitatory or inhibitory effect on the post-neuron. This effectively means that *neurons*, not individual synapses, are either excitatory or inhibitory. However, when looking at synaptic weight matrices $\mathbf{W}$ computed within the NEF, one will find that individual pre-neurons j are both connected excitatorily and inhibitorily to post-neurons (i.e., for some post-neuron i, $w_{ij} > 0$, while for another post-neuron i', $w_{i'j} < 0$).
- **No bias currents.** We assumed that each neuron i possesses a bias current J_i^bias. While this may be seen as a model for the excitability of a neuron (neurons with small J_i^bias need larger input currents), bias currents in the literal sense do not exist in biology.
- **Conductance-based synapses: Post-synaptic currents are not linear.** We defined the post-synaptic current J_i as a linear combination of pre-synaptic activities. If two spikes arrive from a pre-neuron, the total post-synaptic current evoked by these pre-activities will just be the sum of the individual pre-activities. This assumption is also called "current-based synapses", i.e., we are assuming that each synapse is generating a current that is injected into the neuron. Better models of synapses are *conductance-based*, where the amount of current injected into the neuron depends on the momentary membrane potential. This phenomenon can be exploited to compute non-linear multivariate functions without having to represent all variables in the pre-population.

Of course, we made several other modelling assumptions that we do not discuss here. For example, when deriving the dynamics principle, we assumed that the synaptic filter dominates the overall neural dynamics. This is not necessarily true in biology. The neuron itself may have dynamical properties with time-constants significantly longer than the synaptic filter.

Note: *The NEF and biological detail.* As we have discussed several times before, the goal of the NEF is not to build the most biologically realistic network models, but to bridge the gap between neurobiology and behavioural/cognitive models. Correspondingly, extensions to the NEF should either provide a way to incorporate important neurobiological constraints into models (such as Dale's principle), or help explaining higher-level phenomena (such as dedritic computation using conductance-based synapse models).

2 Taking Dale's Principle Into Account and Eliminating Bias Currents

Brain microcircuits mapped out by neurobiologists often distinguish between "excitatory" and "inhibitory" neuron populations. As discussed above, according to Dale's principle, individual neurons can only produce one type of neurotransmitter. Since the effect on the post-neuron (i.e., whether it is excited or inhibited) is mostly determined by the neurotransmitter type, neurons tend to affect their post-neurons either excitatorily or inhibitorily.

With the techniques we have discussed in the lecture so far, it is unclear how to incorporate this constraint into NEF networks. In other words, it is unclear how to force all synaptic weights w_{ij} of a particular pre-neuron j (i.e., columns in the weight matrix $\mathbf{W}$) to be either positive or negative. This makes it harder to build models of the aforementioned brain microcircuits.

Note: *Representation space and synaptic weights.* Remember that we distinguish between the representational space and the neural activity space in the NEF.

Suppose that we wanted to inhibit the post-neuron population, i.e., have synaptic connections w_{ij} that are either zero or negative. Computing an "inhibitory" function in *representation space* such as $f(x) = -x$ does not imply that the neurons in the post ensemble are less active than the neurons in the pre-population (assuming positive x)!

This is partially due to the fact that we typically have "positive" and "negative" encoders in the post-population. Some neurons in the post-population will be more active (i.e., will be excited) when representing smaller values.

Generally speaking, the algebraic sign of the synaptic weights w_{ij} depends on the signs of *both* the decoders $\mathbf{D}$ and the encoders $\mathbf{E}$. Under some circumstances we can choose neural tuning curves of the pre- and post-ensemble in such a way that—depending on the function we compute—weights will be mostly positive or negative. However, ensuring this tends to be quite tricky. The method we discuss in the following is much simpler, although not directly implemented in Nengo.

2.1 Solving for weights in current space

We can incorporate Dale's principle quite easily into NEF networks if we solve for full connection weight matrices $\mathbf{W}$ in "current space". Solving for "full connection weight matrices" $\mathbf{W}$ means that we can no longer use the factorization trick $\mathbf{W} = \mathbf{E}\mathbf{D}$ when simulating our networks, since we never solve for decoders $\mathbf{D}$ in the first place, but directly for $\mathbf{W}$ instead.

In particular, we solve for the synaptic weights of a single post-neuron i. That is, we compute each row $\mathbf{w}_i$ of the weight matrix $\mathbf{W}$ individually. According to the "current-based synapse" model [3], the static (i.e., not taking the synaptic filter into account) post-synaptic current $\hat{J}_i$ of a post-neuron i given the pre activity $\boldsymbol{\alpha}(\mathbf{x})$ of a pre-population representing the value $\mathbf{x}$ is

$$\hat{J}_i = \langle \boldsymbol{\alpha}(\mathbf{x}), \mathbf{w}_i \rangle \,. \tag{1}$$

Furthermore, we have a *normative constraint* from the encoding equation that tells us what input current a neuron should receive if the entire population represents a value $\mathbf{x}$:

$$J_i = \alpha_i \langle \mathbf{x}, \mathbf{e}_i \rangle + J_i^{\mathrm{bias}} \,. \tag{2}$$

Note: Here, *normative constraint* means that, when building a NEF network, we choose parameters α_i, $\mathbf{e}_i$, J_i^{bias} that determine what current *we would like* each individual neuron to receive, if its population as a whole represents a value $\mathbf{x}$. With the method below we solve for connection weights $\mathbf{w}_i$ that approximately compute the current in eq. (2) by linearly decoding the pre-population activities.

When implementing the NEF principle two, i.e., computing a function f in the connection between two ensembles, we need to minimize the error $(\hat{J}_i - J_i)^2$, i.e., we need to make sure that the actual input current $\hat{J}_i$ is equal to the desired input current J_i. Combining eq. (1) and eq. (2) for N samples results in the following optimization problem:

$$\mathbf{w}_i = \arg\min_{\mathbf{w}_i} \sum_{k=1}^{N} \left(\left\langle \mathbf{a}(\mathbf{x}_k), \mathbf{w}_i \right\rangle - \alpha_i \left\langle \mathbf{f}(x_k), \mathbf{e}_i \right\rangle + J_i^{\mathrm{bias}} \right)^2 \,. \tag{3}$$

We can write this in matrix form as

$$\mathbf{w}_i = \arg\min_{\mathbf{w}_i} \left\| \mathbf{A}\mathbf{w}_i - \mathbf{J}^{\mathrm{tar}} \right\|^2 \,,$$

which can be solved using a least-square solver, as we have discussed multiple times before.

Notice that using the resulting weights will approximately evoke the desired target current J_i defined in eq. (2) in the post neuron. This includes the bias current J_i^{bias}, the gain α_i, as well as the encoder $\mathbf{e}_i$. Correspondingly, after solving for weights, the entire network is solely governed by eq. (1). While the neuron parameters J_i^{bias}, α_i, $\mathbf{e}_i$ are still used to control the tuning curves of our neural population (i.e., to establish the "normative constraint", see above), these parameters are no longer part of our simulation. We no longer have to assume that each neuron possesses a bias current J_i^{bias}, this current is implicitly decoded from the pre-population—we have "eliminated the bias current".

Note: *The current translation function.* The function that translates a represented value $\mathbf{x}$ into a current J_i is called "current translation function". So far, we generally chose the affine function given in eq. (2), i.e.,

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

However, note that this function is arbitrary. With the technique detailed above, i.e., solving for weights in current space, we can choose *any* current translation function, and, as a result, *any* tuning curve shape, as long as this function can be decoded from the pre-population. Since decoding affine functions tends to be possible with a small error (see the lecture on analysing representations), this is a sane default.

2.2 Enforcing Dale's Principle

In a sense, the above method of solving for a synaptic weight matrix $\mathbf{W}$ is a reformulation of the decoder computation problem. Instead of computing a single function decoder for the entire post-population, we compute a "current decoder" for each post-neuron. While this allows us to eliminate the bias current from the post-population—since it is implicitly computed as part of the "current decoders"—we are still not taking Dale's principle into account. Luckily, since we now solve for weights directly, this has become much easier.

Consider the general case of the pre-population being split into a set of excitatory and inhibitory neurons. Note that these sets could also be empty, corresponding to the entire neuron population being purely excitatory or inhibitory. Correspondingly, we now have two separate sets of connections: excitatory connections with corresponding weights $\mathbf{w}_i^+$, and inhibitory connections with the corresponding weights $\mathbf{w}_i^-$. We can now rewrite the optimization problem from eq. (3) as

$$\mathbf{w}_i^+, \mathbf{w}_i^- = \arg\min_{\mathbf{w}_i^+, \mathbf{w}_i^-} \left\| \mathbf{A}^+ \mathbf{w}_i^+ - \mathbf{A}^- \mathbf{w}_i^- - \mathbf{J}^{\mathrm{tar}} \right\|^2 , \qquad \text{with respect to} \quad \mathbf{w}_i^+, \mathbf{w}_i^- \geq 0 . \qquad (4)$$

where $\mathbf{A}^+$ and $\mathbf{A}^-$ are the activities of the excitatory and inhibitory pre-neurons. Rearranging this equation yields

$$\mathbf{w}_i' = (\mathbf{w}_i^-, \mathbf{w}_i^+) = \arg\min_{\mathbf{w}_i} \left\| (\mathbf{A}^+, -\mathbf{A}^-)(\mathbf{w}_i^-, \mathbf{w}_i^+)^{\mathrm{T}} - \mathbf{J}^{\mathrm{tar}} \right\|^2 = \arg\min_{\mathbf{w}_i} \left\| \mathbf{A}'(\mathbf{w}_i')^{\mathrm{T}} - \mathbf{J}^{\mathrm{tar}} \right\|^2 ,$$
$$\text{w.r.t. } \mathbf{w}_i' \geq 0 .$$

This problem is called *non-negative least squares* (NNLS), and can, for example, be solved using the SciPy function `scipy.optimize.nnls`.

Using this method, we declare each neuron within an NEF network to be either excitatory or inhibitory, and then solve for connection weight matrices that take this constraint into account. As demonstrated below, depending on the function we are computing, the impact of doing this with respect to the precision is negligible, although there are some idiosyncrasies one has to keep in mind when building networks in this way, particularly the fact that not all functions can be decoded from the pre-activities using purely excitatory/inhibitory weights.

Note: *Multiple pre-populations.* When using the above methods for building networks with multiple pre-populations targeting the same post-population, the bias current must not be decoded from multiple pre-populations at once. There are several ways to accomplish this. For example, when receiving input from n pre-populations, one can just solve for $\frac{1}{n}$ of the bias in each connection.

Another method is to "stack" all pre-population activities into a single vector $\boldsymbol{\alpha}$ to form a "virtual" pre-population, and to compute all connection weights at once. This implicitly distributes the bias current among the individual connections and is exactly the method implemented by `nengo-bio` (see below).

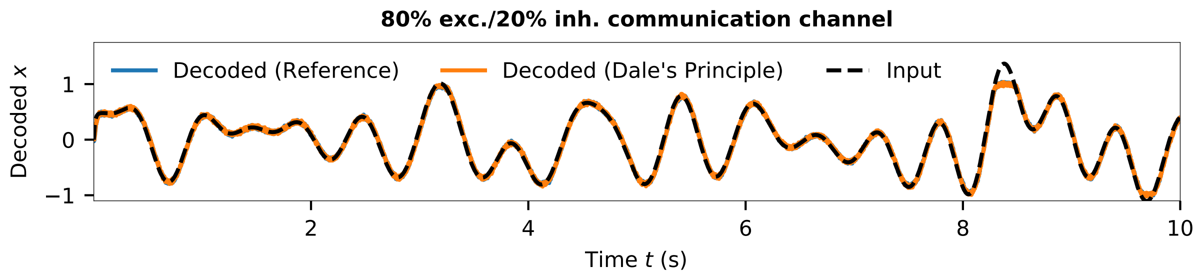


Figure 1: Two communication channels: one using standard (reference) NEF methodology, and one using a pre-population with 80% excitatory neurons and 20% inhibitory neurons. When computing the identity function, there is virtually no loss in accuracy when accounting for Dale's principle. ⌨ *Code*

2.3 Biologically plausible Nengo networks using `nengo-bio`

While we could manually solve for connection weight matrices **W** using the above method and use the resulting connection weight matrices in a "neurons-to-neurons" connection in Nengo, this tends to be quite tedious. An easier way is to use the `nengo-bio`[1] library, an (experimental) free software library extending Nengo to facilitate the construction of biologically (more) plausible neural networks.

Example: An 80% excitatory/20% inhibitory communication channel The following code in `nengo-bio` implements a communication channel from a neuron population with 80% excitatory neurons and 20% inhibitory neurons. This is approximately the ratio of excitatory to inhibitory neurons found in cerebral cortex.

```python
import nengo
import nengo_bio as bio

with nengo.Network(seed=4909) as model:
    nd_in = nengo.Node(nengo.processes.WhiteSignal(
       rms=0.5, high=2.0, period=10.0))

    # Use "bio.Ensemble" objects
    ens_a = bio.Ensemble(n_neurons=101, dimensions=1, p_exc=0.8) # 80% excitatory
    ens_b = bio.Ensemble(n_neurons=102, dimensions=1)

    # Use a nengo.Connection to connect nodes or "normal"
    # nengo.Ensemble objects to a bio.Ensemble
    nengo.Connection(nd_in, ens_a)

    # Use bio.Connection for connections between bio.Ensemble objects
    bio.Connection(ens_a, ens_b)
```

[1] See `https://github.com/astoeckel/nengo-bio`

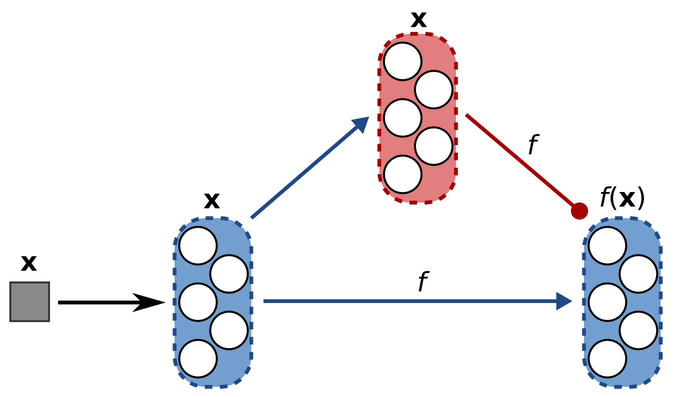
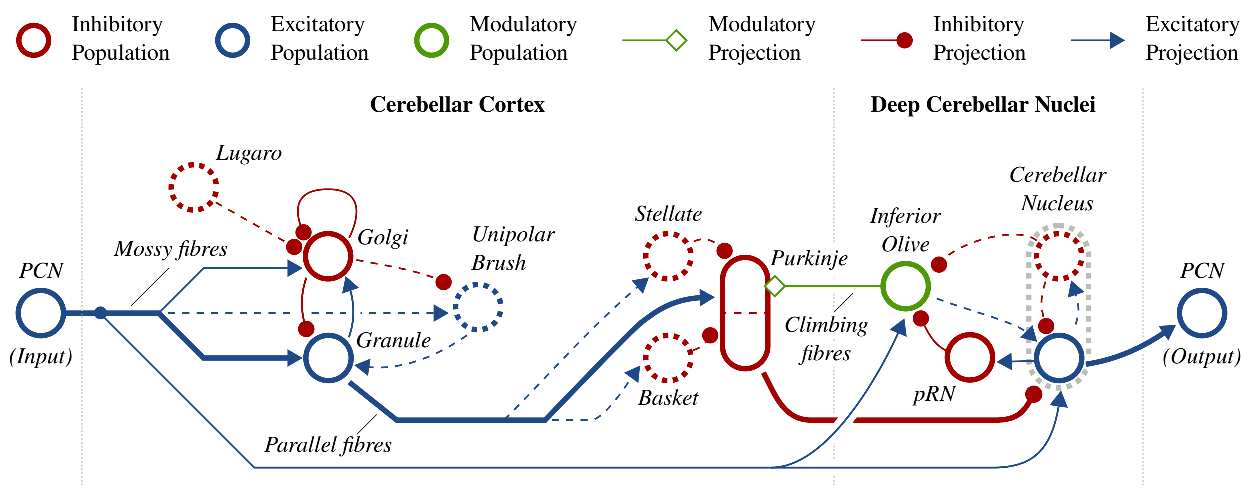


Figure 2: Example of a network with inhibitory interneurons. An excitatory signal is relayed via an inhibitory population of neurons. Blue lines correspond to purely excitatory connections, red lines to purely inhibitory connections. Assuming that both the original neuron population and the interneuron population represent the same value, we can compute a function f across two separate sets of connections.

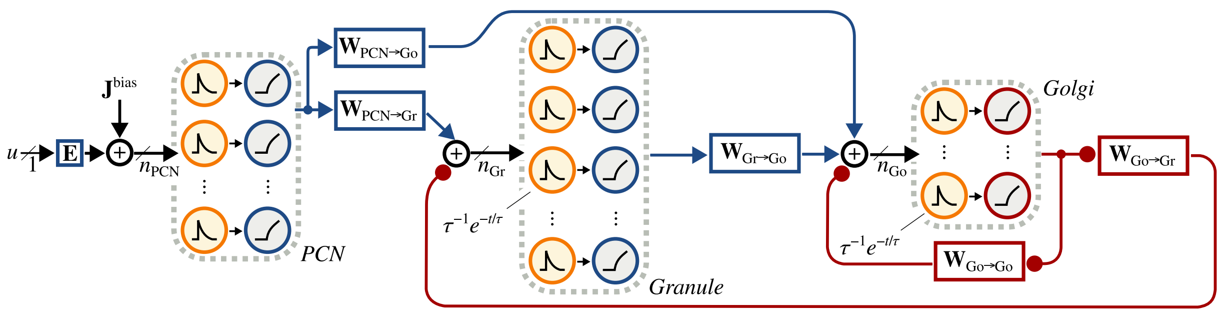
The result of running this code is shown in fig. 1. Restricting the excitatoriness or inhibitoriness of the pre-neurons does not have a visible impact on the accuracy of the system. In fact, the communication channel will still work fine, even for 100% excitatory pre-neurons or 100% inhibitory pre-neurons (given that there is a suitable bias source). Depending on the pre-population tuning curves, the error may increase for more complex functions.

Example: Inhibitory Interneurons Figure 2 depicts a connectivity scheme that is often found in brain microcircuits. A signal $\mathbf{x}$ is relayed from a purely excitatory pre-population via both a direct connection and a population of inhibitory interneurons. Assuming that both pre-populations represent the same value, we can still compute arbitrary functions f across both the excitatory and inhibitory connection. nengo-bio comes with a special syntax for this:

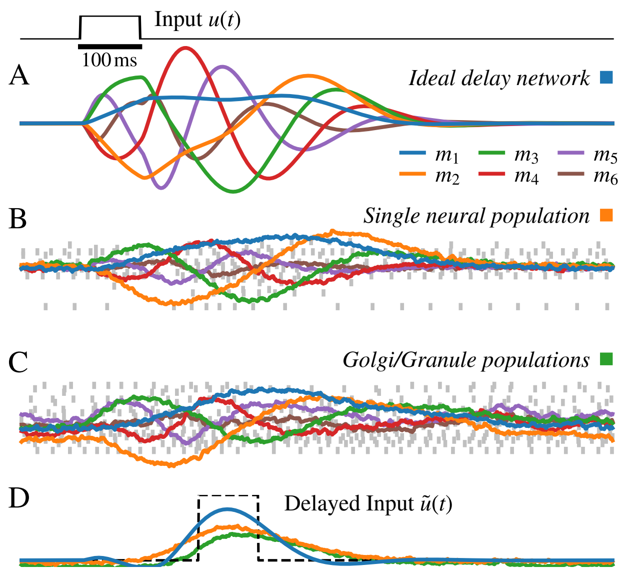
```python
import nengo
import nengo_bio as bio

with nengo.Network(seed=4909) as model:
    nd_in = nengo.Node(nengo.processes.WhiteSignal(
        rms=0.5, high=2.0, period=10.0))

    ens_a = bio.Ensemble(n_neurons=80, dimensions=1, p_exc=1.0)
    ens_b = bio.Ensemble(n_neurons=20, dimensions=1, p_inh=1.0)
    ens_c = bio.Ensemble(n_neurons=100, dimensions=1)

    # Communication channel from A to B
    bio.Connection(ens_a, ens_b)

    # Tell nengo-bio that A and B represent the same value using "set notation"
    bio.Connection({ens_a, ens_b}, ens_c, function=lambda x: x**2)
```



(a) The cerebellar microcircuit



(b) A model of the Golgi-Granule circuit



(c) Diagram showing the delay network response.

Figure 3: Mapping the delay network onto the Golgi-Granule microcircuit. **(a)** Depicts the cerebellar microcircuit, the primary pathway is highlighted in bold. Data from [4, 5]. **(b)** A model of the recurrent Golgi-Granule circuit. Orange circles correspond to the synaptic filters, blue circles to individual neurons. **(c)** Comparing the Golgi-Granule implementations to other delay network representations.

Example: A Model of the Cerebellum The cerebellum (Latin for "little brain") has long been known in the clinical literature to be important for precise motor control. More recent evidence also points at the cerebellum being important for timing and working-memory related cognitive tasks.

The cerbellar microciruitry is one of the most well-studied of all brain substructures (see [4, 5] for details). To summarize, the cerebellum has a predominantly feed-forward architecture (fig. 3a). Signals from neurons in pre-cerebellar nucleii are routed to a layer of so called "granule" or "granular cells". The granular cells are extremely small (hence the name) and possess only few incoming connections. Astonishingly, about half of all neurons in the human brain are cerebellar granule cells. This means that signals from the PCN are mapped onto a vast number of neurons. Granular cells then connect onto Purkinje cells, which in turn connect to the cerebellar nucleus and from there back to other brain regions. The connections between the Granular and Purkinje cells can be modulated (learned) according to "error signals" originating from the Inferior Olive.

Marr proposed in 1969 [6] that this circuit performs pattern extraction in the granular layer and learns to recombine (decode) the extracted patterns into functions such as corrective motor signals (this is also known as the "adaptive filter hypothesis").

One important experiment used to explore cerebellar function is eyeblink conditioning, a form of classical conditioning (see the lecture on learning). The idea is to pair the eyeblink reflex (where a "puff" of air into the eye, the unconditioned stimulus, causes blinking, the unconditioned response) with a conditioned stimulus, such as a tone. Crucially, the tone precedes the puff by a short duration t, typically a few hundred milliseconds. After a while, the person or experimental animal will learn to produce the conditioned response, i.e., will blink even if only the tone is played, while maintaining the original delay. The blink will happen t seconds after the tone. This delayed association is known to be learned by the cerebellum, i.e., the cerebellum is both sufficient and necessary to produce this response.

How exactly the cerebellar microcircuit learns these delays is uncertain. One idea is that the Granule layer somehow produces a temporal function representation, from which the Purkinje cells then decode a delay. While it is—once again—unclear, how exactly this temporal function representation is generated, we have learned in a previous lecture how we can build a spiking neural network that produces such a function representation: the delay network. The granule layer features a set of inhibitory interneurons—the Golgi cells. We could thus try to map the delay network onto this circuit. The resulting circuit is shown in fig. 3b, the response of the circuit compared to more canonical implementations in fig. 3c. This circuit can be easily implemented using `nengo-bio`, taking the excitatoriness and inhibitoriness of individual neuron populations into account.

Note, that in contrast to the standard NEF principle three recurrent architectures, the recurrent connection in the Granular-Golgi subnetwork is through a set of interneurons (the Golgi cells), which potentially changes the dynamics of the system. However, one can mathematically show that exactly the same transformation, i.e., $\mathbf{A}' = \tau\mathbf{A} + \mathbf{I}$ and $\mathbf{B}' = \tau\mathbf{B}$, must be applied to both recurrent and input connections, respectively.

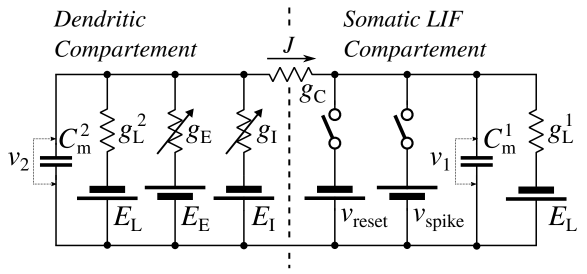


Figure 4: Two-compartment LIF neuron model with conductance-based synapses. Incoming spikes either increase the conductance of the excitatory channel or the inhibitory channel. In the limit of $g_\mathrm{C} \to \infty$, $v_1(t) = v_2(t)$ this model is equivalent to a single-compartment neuron model.

3 Conductance-Based Synapses

We assumed that post-synaptic currents are linear in the pre-synaptic activities, according to the "current-based" synapse model, which implies that every spike arriving at a specific synapse evokes the same post-synaptic current, independent of the neural state. While this is a good first-order approximation, biology tends to be—as always—a little bit more complicated.

3.1 The Conductance-Based Synapse Model

Spikes arriving at a synapse do not directly produce a post-synaptic current. According to Ohm's law, currents J should always be thought of as being evoked by a potential difference U and a conductance $g = 1/R$ that allows charges to flow, i.e., it holds

$$J = \frac{U}{R} = gU\,. \qquad \textit{(Ohm's law)}$$

A neurotransmitter arriving at the post-neuron results in ion-channels gated by that neurotransmitter to open. This results in a change in conductance g of the cell-membrane for ions associated with the corresponding ion-channel. Assume that there are M different ion-channels in the neuron. Each of these ion-channels has an associated "reversal potential E_ℓ". The "conductance-based" synapse model assumes that the conductance for a particular channel ℓ is linear in the pre-synaptic activities $\mathbf{a}^\ell$ of neurons connected to that channel, and that the current J is the product of the conductance and the difference between the membrane potential $v(t)$ and the specific channel reversal potential E_ℓ:

(Conductance-Based Synapse Model)

$$g_\ell(t) = \big(\langle \mathbf{a}^\ell(\mathbf{x}), \mathbf{w}_\ell \rangle * h_\ell\big), \qquad J(t) = \sum_{\ell=1}^{M} g_\ell(t)\big(E_\ell - v(t)\big). \tag{5}$$

The current $J(t)$ is now no longer linear in the pre-synaptic activities—the current depends on the membrane potential $v(t)$, which in return is influenced by the current $J(t)$ according to the neural dynamics. An equivalent circuit diagram for a two-compartment neuron model with conductance-based synapses is depicted in fig. 4.

3.2 Incorporating Synaptic Nonlinearities Into the NEF

As detailed above, the current-based synapse model in eq. (1) assumes that the post-synaptic current J is a linear combination of the pre-synaptic activities $\boldsymbol{\alpha}$. If we assume separate excitatory and inhibitory input "channels", we can rewrite this equation in terms of a function H—the "synaptic nonlinearity model"—that combines these two kinds of inputs, or "channel states". In the case of the current-based synapse model, H is given as

$$\hat{J}_i = H\big(\langle \boldsymbol{\alpha}^+, \mathbf{w}_i^+ \rangle, \langle \boldsymbol{\alpha}^-, \mathbf{w}_i^- \rangle\big) = \langle \boldsymbol{\alpha}^+, \mathbf{w}_i^+ \rangle - \langle \boldsymbol{\alpha}^-, \mathbf{w}_i^- \rangle \,, \tag{6}$$

i.e., the "excitatory" inputs produce a positve, the "inhibitory" inputs a negative current.

When incorporating more complex synapse models into the NEF, the idea is to just use an appropriate function $H(g_1, \ldots, g_M)$ that (nonlinearly) maps channel states (e.g., conductances) onto a current $\hat{J}_i$. We can then obtain synaptic weights by solving a more complex optimization problem similar to eq. (3)

$$\begin{aligned} \mathbf{w}_i^1, \ldots, \mathbf{w}_i^M &= \arg \min_{\mathbf{w}_i^1, \ldots, \mathbf{w}_i^M} \sum_{k=1}^{N} \Big(H\big(\langle \boldsymbol{\alpha}_k^1, \mathbf{w}_i^1 \rangle, \ldots, \langle \boldsymbol{\alpha}_k^M, \mathbf{w}_i^M \rangle\big) - \alpha_i \big\langle \mathbf{f}(x_k), \mathbf{e}_i \big\rangle + J_i^{\text{bias}} \Big)^2 \,, \\ &\text{w.r.t. } w_i^\ell \geq 0 \,, \end{aligned} \tag{7}$$

where the $\boldsymbol{\alpha}_k^\ell$ correspond to the pre-activities of neurons influencing the ℓ-th input channel of the post neuron. As before, k is the sample index out of N samples. Notice that eq. (4) and eq. (7) are exactly the same if H is as defined in eq. (6).

3.3 Finding the Synaptic Nonlinearity H

The optimization problem in eq. (7) tells us how to solve for synaptic weights in case we are using a more complex synapse model summarized by H. However, this is fairly abstract, and we do not really know how to determine this function H.

The idea is that we can measure the output spike rate for given, constant channel states. That is, we have a multivariate neural response function $\mathcal{G}[g_1, \ldots, g_M]$ that maps the channel states onto a spike rate. We can then decompose this function into a one-dimensional neural response function $G[J]$ and the synaptic nonlinearity H, i.e.,

$$\mathcal{G}[g_1, \ldots, g_M] = G[H(g_1, \ldots, g_M)] \Leftrightarrow H(g_1, \ldots, g_M) = J \Leftrightarrow G[J] = \mathcal{G}[g_1, \ldots, g_M] \,.$$

In order to find an H that (approximately) fulfils the above relationship, we can parameterise H, empirically measure $\mathcal{G}[g_1, \ldots, g_M]$, and fit the model parameters to the currents J corresponding to the output rate according to a simple LIF model $G[J]$. In other words, we apply the inverse of $G^{-1}[J]$ to the measured output rates, giving us the LIF-equivalent output current J for each set of channel state $g_1, \ldots, g_M$ (fig. 5). We then fit the parameters of H such that H approximately maps the channel states onto the reconstructed currents. For the more complex neuron model in fig. 4, a suitable H tends to be

$$H(g_\mathrm{E}, g_\mathrm{I}) = \frac{b_1 g_\mathrm{E} + b_2 g_\mathrm{I}}{a_0 + a_1 g_\mathrm{E} + a_2 g_\mathrm{I}} \,, \qquad \text{where} \quad g_\mathrm{E}, g_\mathrm{I}, a_0, a_1, a_2 \geq 0 \,. \tag{8}$$

Note that we can still phrase eq. (7) as a simple NNLS problem for this particular H.

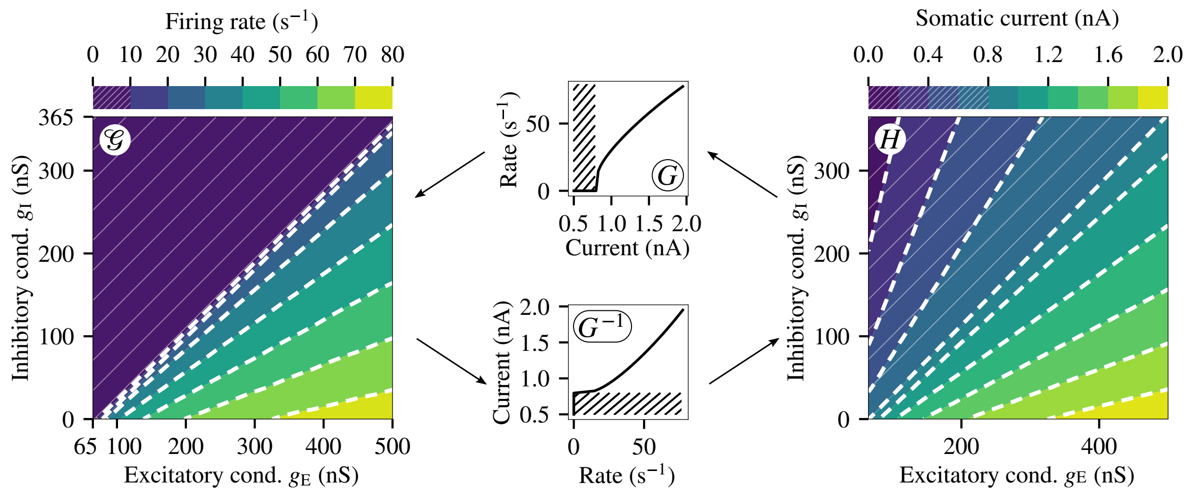


Figure 5: Illustration of the idea of converting a multivariate response function $\mathcal{G}(g_\mathrm{E}, g_\mathrm{I})$ into a synaptic nonlinearity model H by applying the inverse G^{-1} if a simple neural response function. The functions are ill-defined for zero output rates (hatched regions).

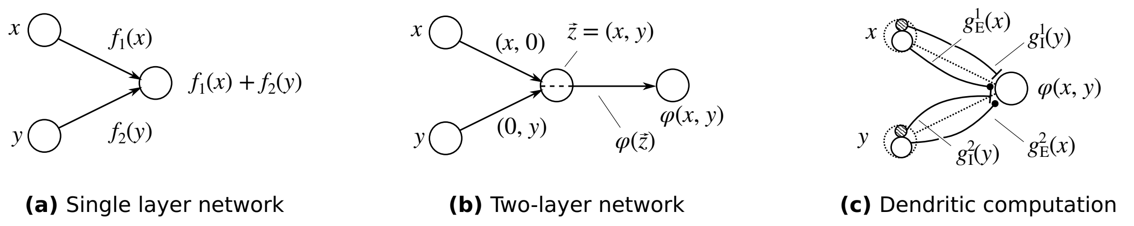


Figure 6: Network architectures for computing multivariate functions. **(a)** If the input variables are represented in different neuron populations, we can only compute a linear combination of functions of these individual variables. **(b)** In a "two-layer network" with an intermediate representation, we can compute arbitrary functions of the two variables. **(c)** Dendritic computation allows us to compute nonlinear multivariate functions by exploiting synaptic nonlinearities.

3.4 Computing Nonlinear Multivariate Functions

At the beginning of this lecture we noted that we should refrain from adding complexity to the NEF for complexity's sake, and instead think about the role of biological detail in terms of the computation, behaviour, or function that is being performed. So, why should we care about incorporating a more complex synapse model into the NEF?

The answer is that by taking the nonlinear effects in the synapses, or, the "dendritic" region of the neuron into account, we can use this detail to compute more complex functions in simple networks. This is also known as "dendritic computation" [7] and is implemented in `nengo-bio`.

Remember that it is not possible to compute a nonlinear, multivariate function such as multiplication if the inputs are not represented in the same pre-population. A network such as fig. 6a cannot compute the product of two numbers, whereas the network in fig. 6b can.

Using the nonlinearity model H from eq. (8) for the two-compartment neuron model with conductance-based synapses in fig. 4, we can compute multiplication with a relatively small

Table 1: Exploring dendritic computation for the three networks in fig. 6. Given values are the average normalised RMSE over 256 experiments with 100 neruons per population (200 in the intermediate population for setup **(b)**). Maximum firing rates between 50 Hz to 100 Hz. See [1] for details.

	Experiment setup						
	Standard LIF			Two comp. LIF $g_C = 50\,\mathrm{nS}$		Two comp. LIF $g_C = 100\,\mathrm{nS}$	
Target	no relaxation	Ⓐ standard	Ⓑ two-layer	Ⓒ standard	noise model	standard	noise model
$x + y$	$5.1 \pm 0.6\%$	$5.5 \pm 1.1\%$	$11.0 \pm 1.3\%$	**$3.2 \pm 1.1\%$**	$9.1 \pm 1.2\%$	$5.1 \pm 1.2\%$	$11.5 \pm 1.3\%$
$x \times y$	$26.2 \pm 0.4\%$	$21.5 \pm 6.6\%$	$15.4 \pm 4.0\%$	$13.9 \pm 2.9\%$	**$11.9 \pm 1.8\%$**	$18.2 \pm 4.0\%$	$14.3 \pm 2.1\%$
$\sqrt{x \times y}$	$14.1 \pm 0.4\%$	$19.7 \pm 6.1\%$	$16.3 \pm 3.0\%$	$9.7 \pm 2.6\%$	**$7.1 \pm 1.0\%$**	$13.3 \pm 4.2\%$	$8.9 \pm 1.7\%$
$(x \times y)^2$	$44.5 \pm 0.6\%$	$33.0 \pm 6.6\%$	**$18.7 \pm 6.7\%$**	$27.7 \pm 4.1\%$	$27.4 \pm 4.1\%$	$34.3 \pm 5.3\%$	$30.3 \pm 4.3\%$
$x/(1+y)$	$6.0 \pm 0.4\%$	$5.2 \pm 0.7\%$	$9.5 \pm 0.8\%$	**$3.4 \pm 1.0\%$**	$10.0 \pm 1.6\%$	$5.3 \pm 1.3\%$	$14.0 \pm 1.9\%$
$\|(x,y)\|$	$8.0 \pm 0.4\%$	$5.7 \pm 1.1\%$	$10.5 \pm 1.0\%$	**$3.1 \pm 1.3\%$**	$8.9 \pm 1.2\%$	$4.3 \pm 1.8\%$	$12.3 \pm 1.8\%$
$\mathrm{atan}(x,y)$	$10.3 \pm 0.3\%$	$8.6 \pm 1.0\%$	$13.4 \pm 1.1\%$	**$5.8 \pm 1.3\%$**	$8.4 \pm 1.0\%$	$7.0 \pm 1.2\%$	$12.7 \pm 1.6\%$
$\max(x,y)$	$14.9 \pm 0.3\%$	$10.0 \pm 0.9\%$	$11.3 \pm 1.4\%$	**$5.5 \pm 0.9\%$**	$7.7 \pm 0.9\%$	$7.3 \pm 0.9\%$	$9.7 \pm 1.0\%$

error over the domain $(x, y) \in [0, 1]^2$, as well as a wide variety of other nonlinear multivariate functions. Table 1 shows the results from an experiment demonstrating that dendritic computation can even be better than a two-layer network for a wide variety of functions. See [1] for more detail.

References

[1] Andreas Stöckel and Chris Eliasmith. "Passive Nonlinear Dendritic Interactions as a General Computational Resource in Functional Spiking Neural Networks". In: *arXiv* (2019). eprint: `arXiv:1904.11713`.

[2] Piergiorgio Strata and Robin Harvey. "Dales Principle". In: *Brain Research Bulletin* 50.5 (1999), pp. 349–350. ISSN: 0361-9230. DOI: `https://doi.org/10.1016/S0361-9230(99)00100-8`. URL: `http://www.sciencedirect.com/science/article/pii/S0361923099001008`.

[3] Arnd Roth and Mark C. W. van Rossum. "Modeling Synapses". In: *Computational Modeling Methods for Neuroscientists*. Ed. by Erik De Schutter. The MIT Press, 2009, pp. 139–159.

[4] Masao Ito. "Cerebellar Cortex". In: *Handbook of Brain Microcircuits*. Ed. by Gordon Shepherd and Sten Grillner. 1st. Oxford, UK: Oxford University Press, Aug. 2010, pp. 293–300. ISBN: 978-0-19-538988-3. URL: `https://oxfordmedicine.com/view/10.1093/med/9780195389883.001.0001/med-9780195389883-chapter-028`.

[5] Rodolfo R. Llinás. "Olivocerebellar System". In: *Handbook of Brain Microcircuits*. Ed. by Gordon Shepherd and Sten Grillner. 1st. Oxford, UK: Oxford University Press, Aug. 2010, pp. 301–308. ISBN: 978-0-19-538988-3. URL: `https://oxfordmedicine.com/view/10.1093/med/9780195389883.001.0001/med-9780195389883-chapter-029`.

[6] D Marr. "A Theory of Cerebellar Cortex". In: *The Journal of physiology* 202.2 (June 1969), pp. 437–470. ISSN: 0022-3751. DOI: `10.1113/jphysiol.1969.sp008820`. PMID: 5784296. URL: `https://www.ncbi.nlm.nih.gov/pubmed/5784296`.

[7] Michael London and Michael Häusser. “Dendritic Computation”. In: *Annual Review of Neuroscience* 28.1 (2005), pp. 503–532. DOI: `10.1146/annurev.neuro.28.061604.135703`. pmid: 16033324. URL: `https://doi.org/10.1146/annurev.neuro.28.061604.135703`.