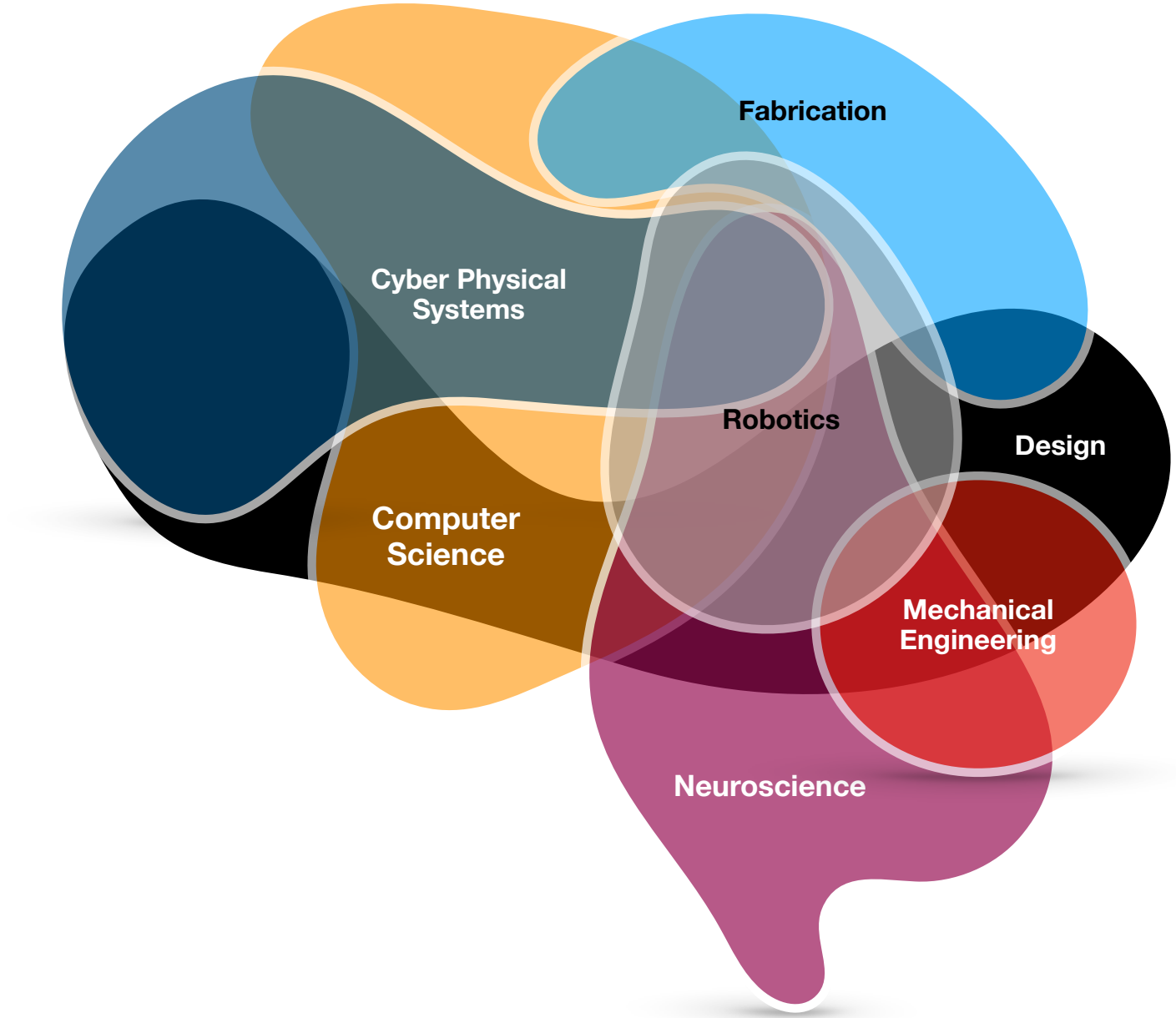
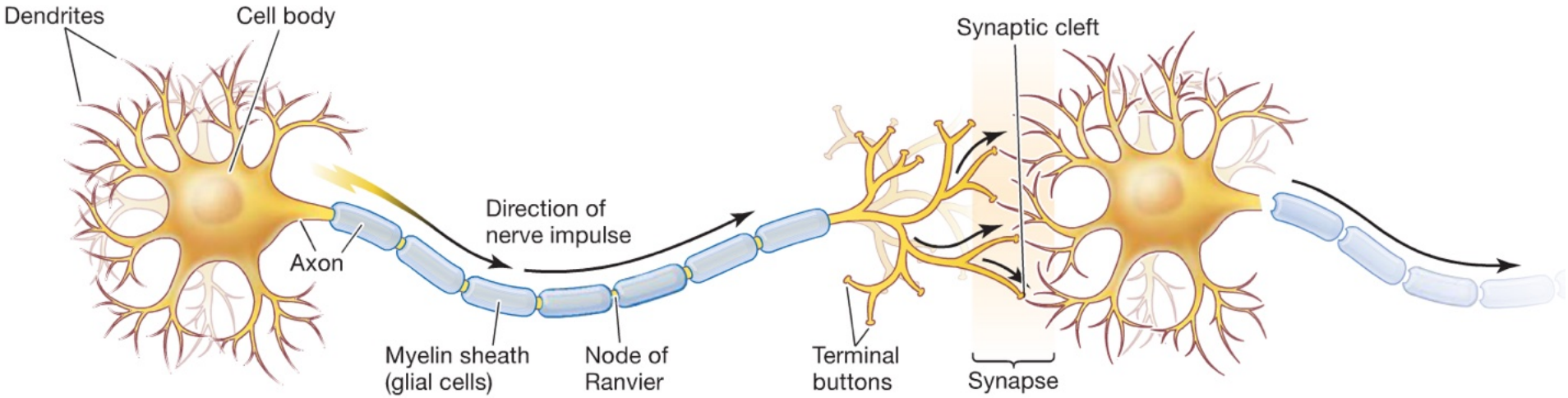
NEL

NEURO-BIOMORPHIC ENGINEERING LAB



The computational building blocks of the brain



Main thesis - the neuron doctrine:

- The **soma** (site of signals integration), **dendrites** (synapses from source neurons) and **axon** (output pathway on target neurons)
- Neurons and synapses can be described by relatively compact, functional, phenomenological **mathematical models**.
- Neuronal communication can be summarized in binary, asynchronous messages (**spikes**).

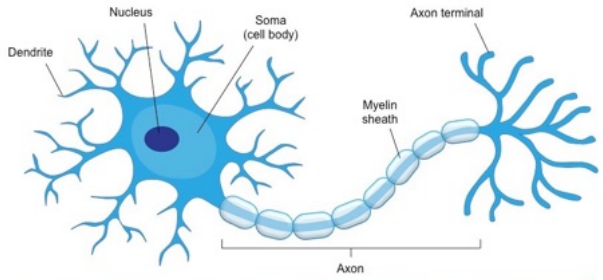
Models of Neuronal Dynamics

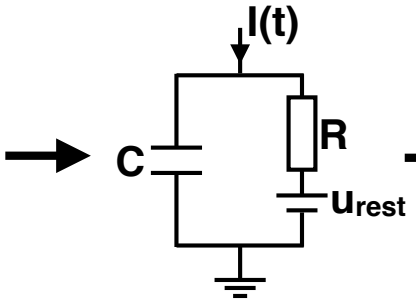
The passive membrane model

The simplest - most used electrical-mathematical model for neuronal dynamics

Capacitance for modeling passive membrane behavior - separation of ions. Membrane acts as a leaky capacitor whose voltage, in the absence of injected current, decays (or “leaks”) to a resting voltage level.

Resistance for modeling ion channels



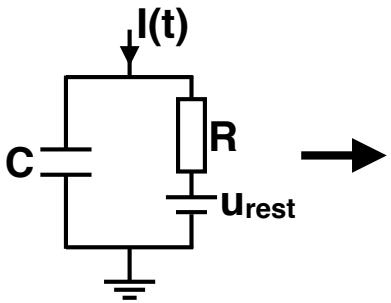


$$\tau \frac{d}{dt} V(t) = -V(t) + R \cdot I(t)$$

see recorded session

Models of Neuronal Dynamics

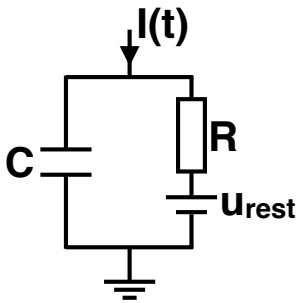
The passive membrane model



$$\tau \frac{d}{dt} V(t) = -V(t) + R \cdot I(t)$$

Models of Neuronal Dynamics

The passive membrane model



$$\tau \frac{d}{dt} V(t) = -V(t) + R \cdot I(t)$$

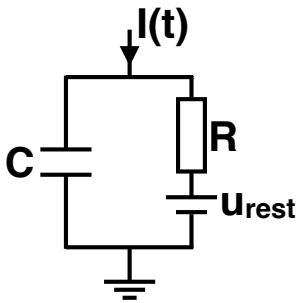
see recorded session

step current solution

$$u(t) = u_{rest} + R \cdot I_0 \left[1 - e^{-\frac{t - t_0}{\tau}} \right]$$

Models of Neuronal Dynamics

The passive membrane model



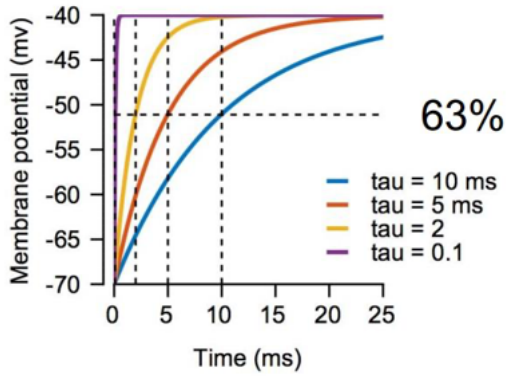
$$\tau \frac{d}{dt} V(t) = -V(t) + R \cdot I(t)$$

see recorded session

step current solution

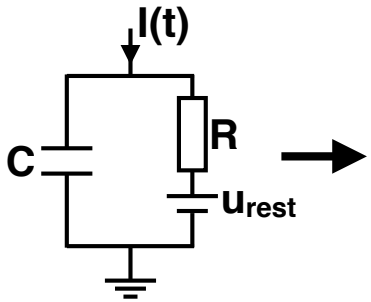
$$u(t) = u_{rest} + R \cdot I_0 \left[1 - e^{-\frac{t-t_0}{\tau}}\right]$$

=RC, which is the time it took for the membrane potential do developed to 63% of its maximum value (I(t) • R)



Models of Neuronal Dynamics

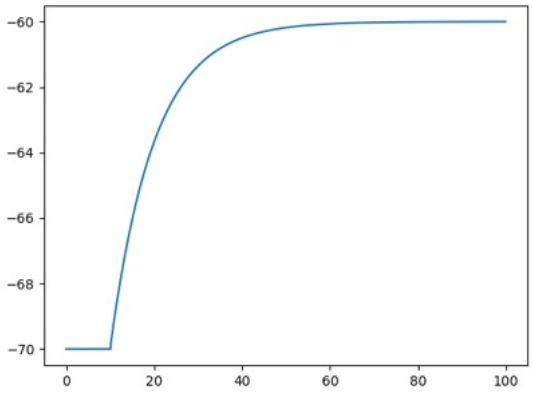
The passive membrane model

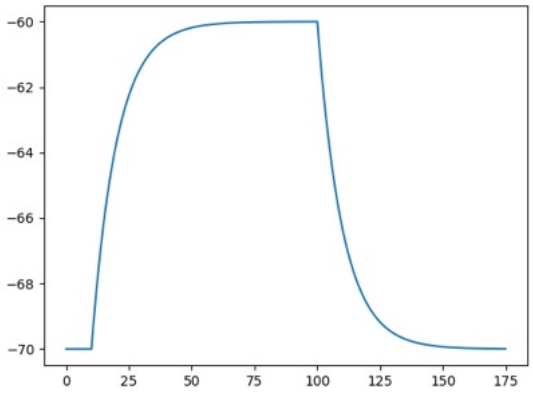


$$\tau \frac{d}{dt} V(t) = -V(t) + R \cdot I(t)$$

Solutions for **flat input**, **step current**, **pulse, short pulse** and **arbitrary input** are given in the lecture's handout

Membrane voltage reacts to a step current stimulation

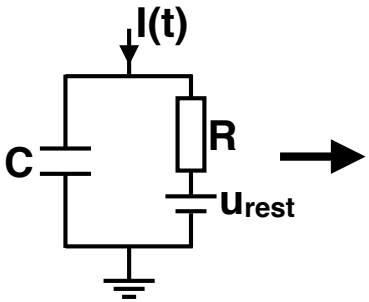




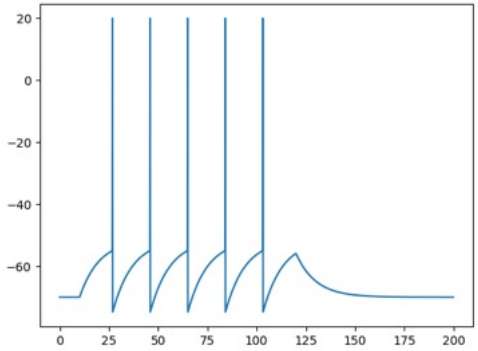
Membrane voltage reacts to a pulse current stimulation

Models of Neuronal Dynamics

The integrate and fire model



$$\tau \frac{d}{dt} V(t) = -V(t) + R \cdot I(t)$$



- Adding a **threshold behavior**, in which a neuron releases a **spike** of voltage and then **reseted** to a new u_r value, if the membrane voltage reaches a value **θ**.
- There is **no explicit modeling** of the ion channel kinetics responsible for this **spiking**. Since all action potentials sent down the axon are to a good approximation identical, the exact shape of the action potential is not important. The only informative feature of a neuron's spiking is their **timing**.
- For an input step function, the **largest** the current is, the **faster** the membrane voltage will reach θ. Therefore **firing frequency** is dependent upon the input current. This frequency-current relation is known as the '**f-I curve**'.

Level of abstraction in conceptualizing and modeling the brain

High levels of abstraction

The Cat is Out of the Bag: Cortical Simulations with 10^9 Neurons, 10^{13} Synapses

Rajagopal Ananthanarayanan[1], Steven K. Esser[1]
Horst D. Simon[2], and Dharmendra S. Modha[1]

[1]IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120
[2]Lawrence Berkeley National Laboratory, One Cyclotron Road, Berkeley, CA 94720
{ananthr,sesser}@us.ibm.com, hdsimon@lbl.gov, dmodha@us.ibm.com

In the quest for cognitive computing, we have built a **massively parallel cortical simulator**, C2, that incorporates a number of innovations in computation, memory, and communication. Using C2 on LLNL's Dawn Blue Gene/P supercomputer with **147,456 CPUs** and **144 TB** of main memory, we report two cortical simulations – at unprecedented scale – that effectively saturate the entire memory capacity and refresh it at least every simulated second. The simulation consists of **1.6 billion neurons** and **8.87 trillion synapses** with experimentally-measured gray matter thalamocortical connectivity. We demonstrate nearly perfect weak scaling and attractive strong scaling. The simulations, which incorporate phenomenological spiking neurons, individual learning synapses, axonal delays, and dynamic synaptic channels, exceed the scale of the cat cortex, marking the dawn of a new era in the scale of cortical simulations.

NOEL

Neuroscience Expert Dr. Henry Markram on the IBM “Cat Brain” Simulation: “IBM’s claim is a HOAX”

Shortly after IBM announced their cat-scale brain simulation, Henry Markram of the Blue Brain Project published a very strong criticism of the claim. He called it *"**a mega public relations stunt - a clear case of scientific deception of the public**"*. According to Markram these simulations do not come close to the complexity of an ant brain, let alone that of a cat brain.

Markram's first argument was that although the number of simulated neurons roughly equals that of a cat brain, the **model** used for each individual neuron was **trivially simple**. The neurons were modeled as single compartment "dots" completely lacking in biological realism. **Genuine simulation** of real neurons requires solving **millions of times more equations** than were used by IBM. Thus, **not even a millionth of a cat brain was simulated**.

NOEL

Models of Neuronal Dynamics

The Izhikevich neuron model

- A quadratic integrate-and-fire type model with a recovery variable.
- It can replicate several characteristics of biological neurons while remaining computationally efficient.

$$\frac{dv}{dt} = 0.04v^2 + 5v + 140 - u + I$$
$$\frac{du}{dt} = a(b \cdot v - u)$$

- v represents the membrane potential of the neuron
- u represents a membrane recovery variable, which accounts for the activation of K+ and inactivation of Na+ ionic currents.
- In contrast to LIF, it was designed to fit biological observable behavior: the expression $0.04v^2+5v+140$ was obtained by fitting the spike initiation dynamic of a cortical neuron

Models of Neuronal Dynamics

The Izhikevich neuron model

$$\frac{dv}{dt} = 0.04v^2 + 5v + 140 - u + I$$

$$\frac{du}{dt} = a(b \cdot v - u)$$

$$\text{if } v \geq 30\,\text{mV}, \text{ then } \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases}$$

- Parameter a describes the time scale for membrane recovery.
- Parameter b describes the sensitivity of the membrane recovery to fluctuations in membrane potential, allowing achieving sub-threshold spiking.
- Parameter c commonly describes post-spike membrane potential (.
- Parameter d describes the membrane post-spike recovery.

Why is it such a powerful model?

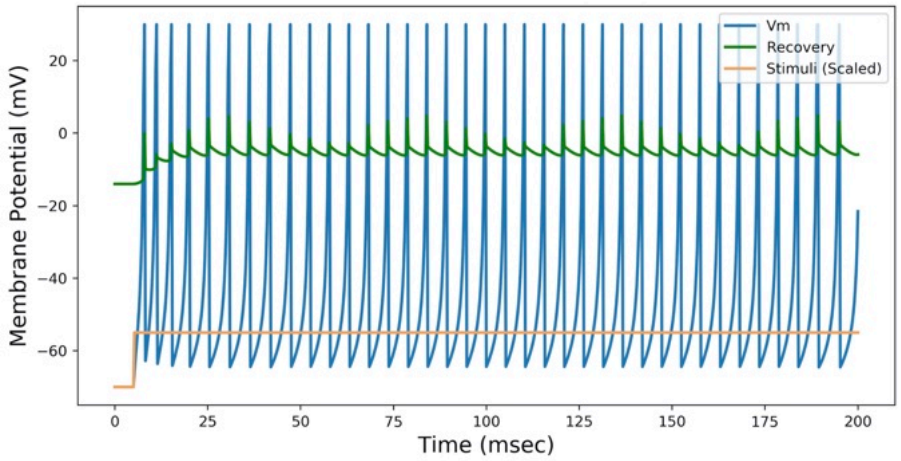
NOEL

Models of Neuronal Dynamics

The Izhikevich neuron model

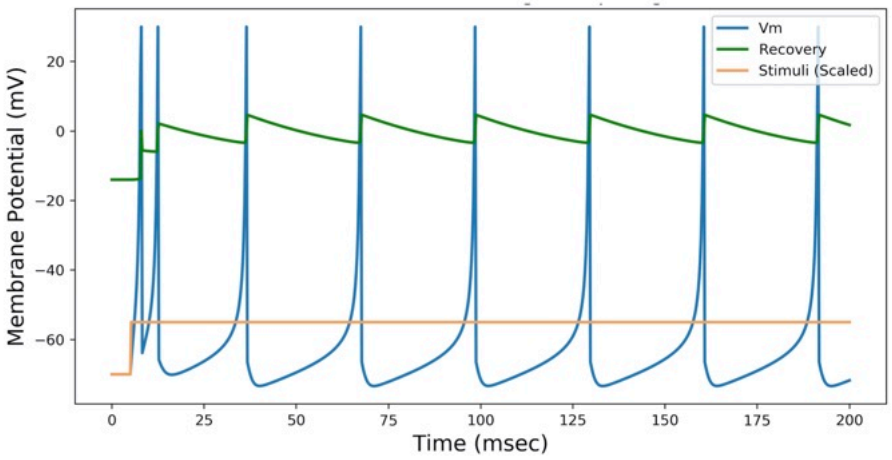
- Different value combination of these parameters account for different neuronal dynamic
- This model is frequently used in large scale simulations, such as the once performed by IBM.

Fast spiking



a = 0.1, b = 0.2, c = –65, d = 2

Regular spiking



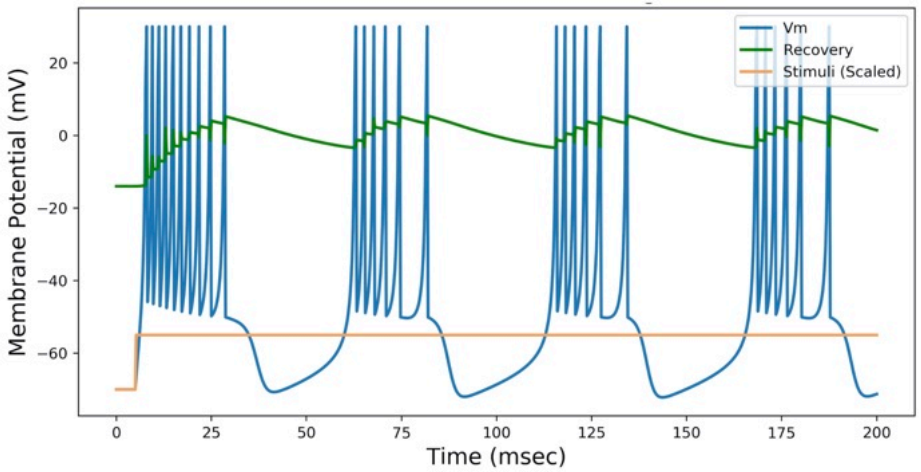
a = 0.02, b = 0.2, c = –65, d = 8

Models of Neuronal Dynamics

The Izhikevich neuron model

- Different value combination of these parameters account for different neuronal dynamic
- This model is frequently used in large scale simulations, such as the once performed by IBM.

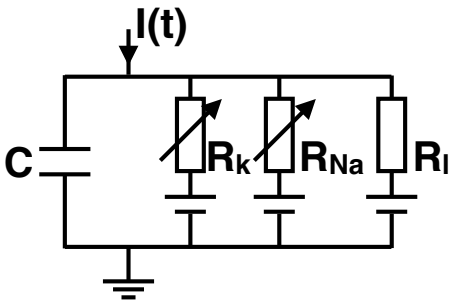
Chattering



a = 0.02, b = 0.2, c = –50, and d = 2

Models of Neuronal Dynamics

The Hodgkin-Huxley model



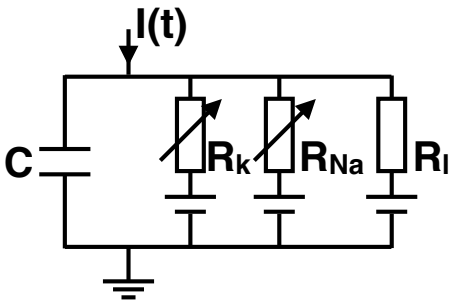
$$C\frac{d}{dt}u = \underbrace{-g_{Na}m^3h(u-E_{Na})}_{I_{Na}} \underbrace{-g_k n^4(u-E_K)}_{I_K} \underbrace{-g_l(u-E_l)}_{I_{leak}} \underbrace{-I(t)}_{stimulus}$$

The most famous **biophysically based neuron model:** It models the voltage- and time-dependent conductances of real ion channels

Was originally used to describe the dynamics of the action potential in the squid giant axon but the description of the conductances in terms of activation and inactivation variables is now used ubiquitously to model neurons in a wide range of species and brain areas.

Models of Neuronal Dynamics

The Hodgkin-Huxley model



gate conductance gating variables (n, m, h)

$$C\frac{d}{dt}u = -g_{Na}m^3h(u-E_{Na}) - g_k n^4(u-E_K) - g_l(u-E_l) - I(t)$$

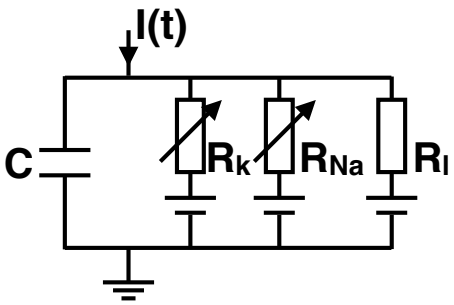
Ion reverse potential

Gating variables:

$$\frac{d}{dt}m = -\frac{m-m_0(u)}{\tau_m(u)}$$

Models of Neuronal Dynamics

The Hodgkin-Huxley model



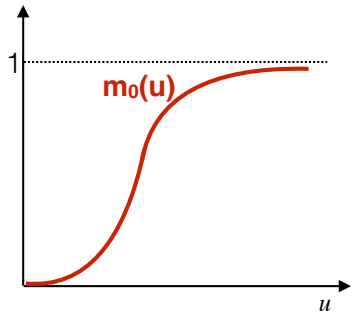
gate conductance gating variables (n, m, h)

$$C\frac{d}{dt}u = -g_{Na}m^3h(u-E_{Na}) - g_k n^4(u-E_K) - g_l(u-E_l) - I(t)$$

Ion reverse potential

Gating variables:

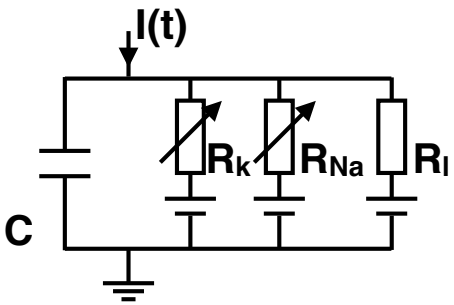
$$\frac{d}{dt}m = -\frac{m-m_0(u)}{\tau_m(u)} \rightarrow m(t) = m_0(u_1) + [m_0(u_0) - m_0(u_1)]\exp\left[\frac{-(t-t_0)}{\tau_m(u_1)}\right]$$



When the voltage is high, m is approaching m_0=1 and the gate opens. Values are derived experimentally.

Models of Neuronal Dynamics

The Hodgkin-Huxley model



$$C\frac{d}{dt}u = -g_{Na}m^3h(u-E_{Na}) - g_k n^4(u-E_K) - g_l(u-E_l) - I(t)$$

gate conductance

gating variables (n, m, h)

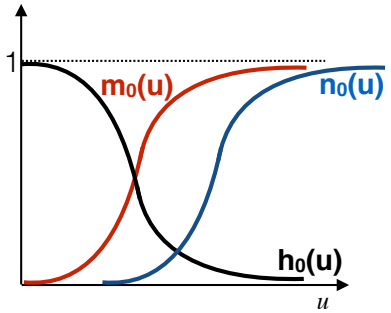
Ion reverse potential

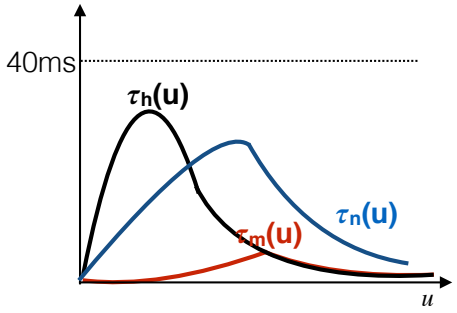
Gating variables:

$$\frac{d}{dt}m = -\frac{m-m_0(u)}{\tau_m(u)} \rightarrow m(t) = m_0(u_1) + [m_0(u_0) - m_0(u_1)]\exp\left[\frac{-(t-t_0)}{\tau_m(u_1)}\right]$$

$$\frac{d}{dt}n = -\frac{n-n_0(u)}{\tau_n(u)}$$

$$\frac{d}{dt}h = -\frac{h-h_0(u)}{\tau_n(u)}$$

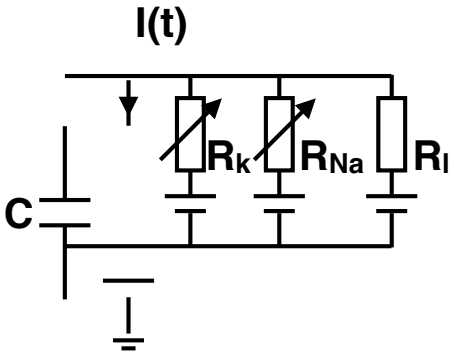




A general mathematical model for Ion channels

Models of Neuronal Dynamics

The Hodgkin-Huxley model



gate conductance gating variables (n, m, h)

$$C\frac{d}{dt}u = -g_{Na}m^3h(u-E_{Na}) - g_k n^4(u-E_K) - g_l(u-E_l) - I(t)$$

Ion reverse potential

Gating variables:

$$\frac{d}{dt}m = -\frac{m-m_0(u)}{\tau_m(u)} = a_m(u)(1-m) - \beta_m m$$

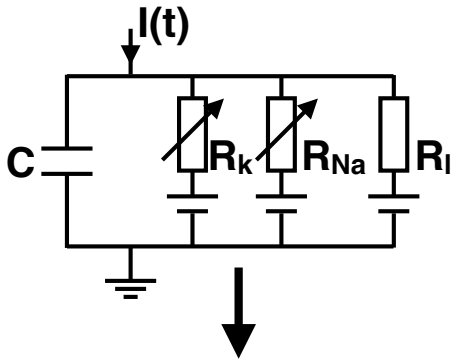
$$\frac{d}{dt}n = -\frac{n-n_0(u)}{\tau_m(u)} = a_n(u)(1-n) - \beta_n n$$

$$\frac{d}{dt}h = -\frac{h-h_0(u)}{\tau_m(u)} = a_h(u)(1-h) - \beta_m h$$

x	E_x [mV]	g_x [mS / cm^2]
Na	55	40
K	-77	35
L	-65	0.3

x	$\alpha_x(u\ /\ \text{mV})$ [ms^{-1}]	$\beta_x(u\ /\ \text{mV})$ [ms^{-1}]
n	$0.02\,(u-25)\ /\ \left[1-e^{-(u-25)/9}\right]$	$-0.002\,(u-25)\ /\ \left[1-e^{(u-25)/9}\right]$
m	$0.182\,(u+35)\ /\ \left[1-e^{-(u+35)/9}\right]$	$-0.124\,(u+35)\ /\ \left[1-e^{(u+35)/9}\right]$
h	$0.25\,e^{-(u+90)/12}$	$0.25\,e^{(u+62)/6}/e^{(u+90)/12}$

The Hodgkin-Huxley model

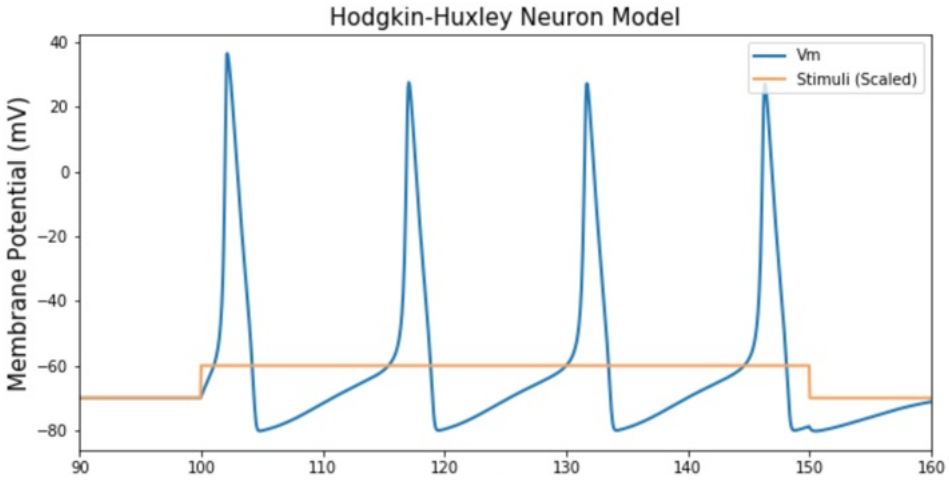


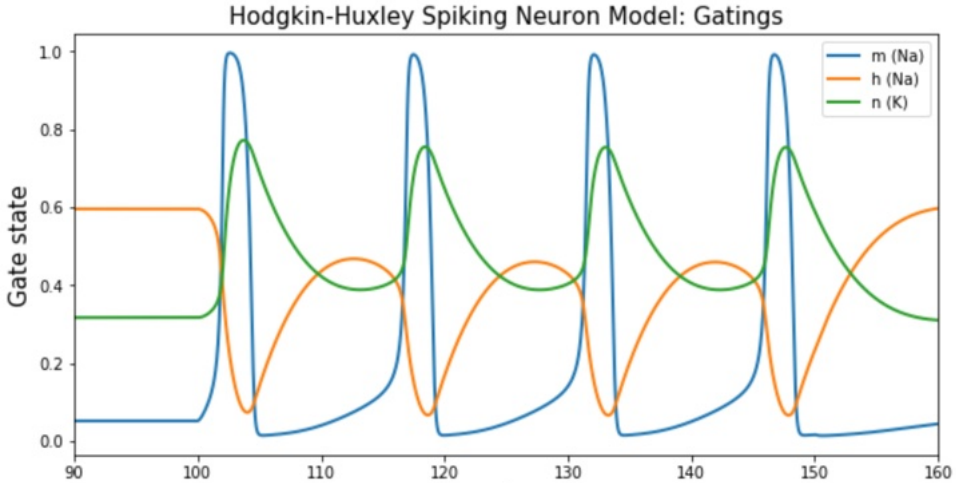
$$C\frac{d}{dt}u = -g_{Na}m^3h(u-E_{Na}) - g_k n^4(u-E_K) - g_l(u-E_l) - I(t)$$

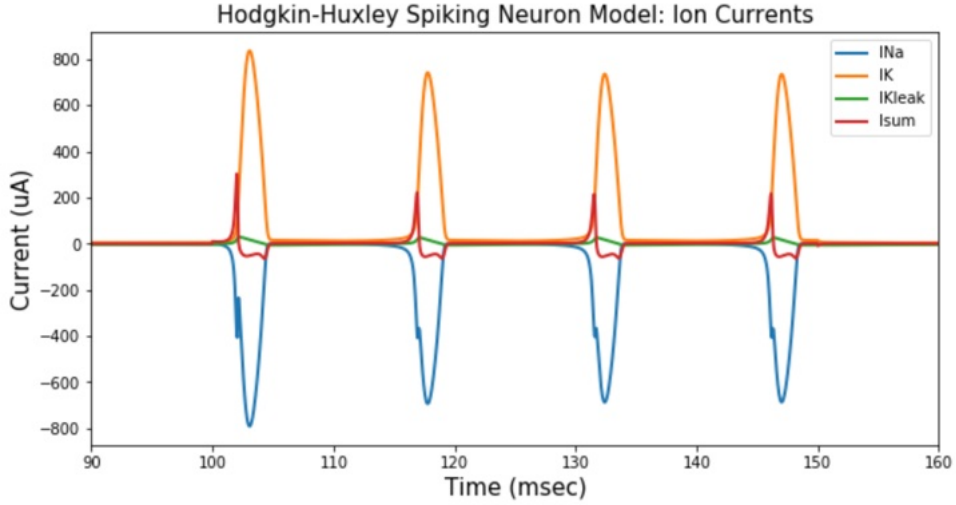
$$\frac{d}{dt}m = -\frac{m-m_0(u)}{\tau_m(u)}$$

$$\frac{d}{dt}n = -\frac{n-n_0(u)}{\tau_m(u)}$$

$$\frac{d}{dt}h = -\frac{h-h_0(u)}{\tau_m(u)}$$







Cell

Reconstruction and Simulation of Neocortical Microcircuitry

It is known that **different types of neurons** are connected through synapses with **different dynamics** and **strengths**, **strategically positioned** at different locations on the neurons' dendrites, somata, and axons, but the **functional significance** of this organization remains **unclear.**

Computational approaches that **abstract away** this level of bio-logical detail have **not been able to explain** the **functional significance** of such intricate cellular and synaptic organization.

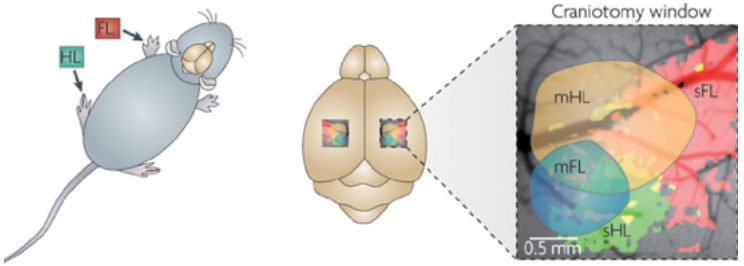
Although **future experimental research** will undoubtedly **advance our knowledge**, it is **debatable** whether experimental mapping **alone** can provide enough data to answer these questions.

Here, we present a **complementary algorithmic approach** that reconstructs neuronal microcircuitry across all layers using available sparse data and that **leverages biological principles** and **interdependencies** between datasets to **predict missing biological data**.

Cell

Reconstruction and Simulation of Neocortical Microcircuitry

We **digitally reconstructed** a small volume of tissue from layers 1 to 6 of the hind-limb somatosensory cortex of 2-week-old Wistar rat.



This model system was chosen not only because it is one of the most **comprehensively characterized** in the neocortex, but also because **experimental data** on its cellular and synaptic organization are **readily available** and **validation experiments** are relatively easy to perform.

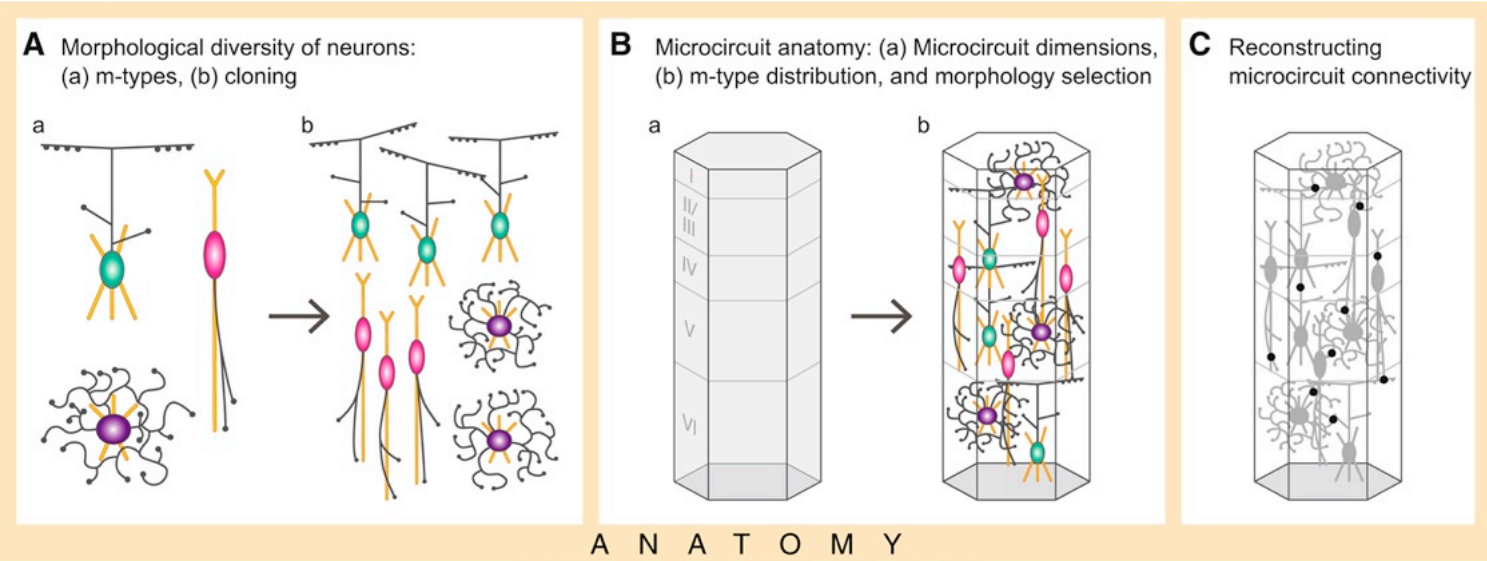
Neuro & Biomorphic Engineering LAB

Reconstruction of anatomical properties

We recorded and **digitally reconstructed neurons** from in vitro brain slices and **classified** the neurons in terms of established **morphological types**

We **positioned** the neurons in a digital volume of objectively defined dimensions according to **experimentally based estimates** of their **layer specific densities**

We reconstructed the connectivity between the neurons



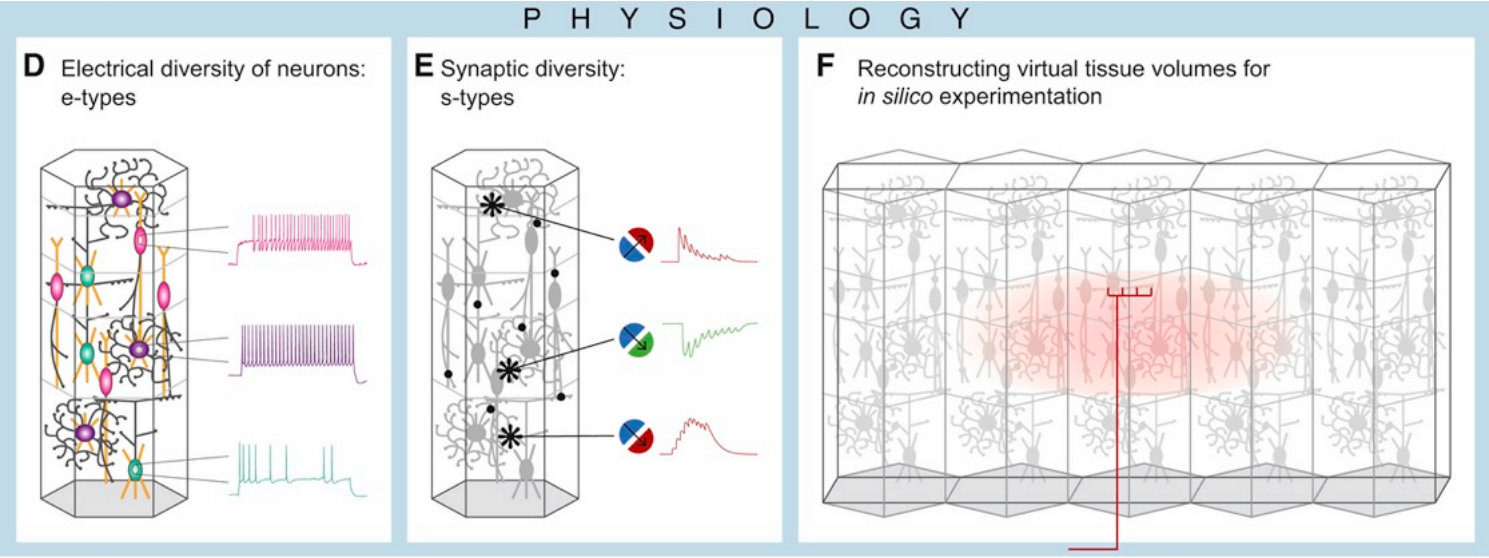
Cell

Reconstruction of physiological properties

Neurons were classified into **electrical types** and models were produced that captured the characteristic electrical behavior of each type.

Synapses were modeled to capture the characteristic **synaptic dynamics** of particular synapse types

We constructed a **virtual slice** and reconstructed **thalamic input** using experimental data



Neuro & Biomorphic Engineering LAB

Cell

Reconstruction and Simulation of Neocortical Microcircuitry

This approach yielded a **first-draft digital reconstruction of the microcircuitry**, which was **validated** against a multitude of experimental datasets **not used** in the reconstruction.

The results suggest that **it is possible** to obtain dense maps of neural microcircuitry **without measuring every conceivable biological parameter** and point to minimal datasets required, i.e., **strategic data**.

Integrating complementary, albeit sparse, datasets also makes it possible to reconcile discrepancies in the literature, at least partially addressing the problem of data quality and reproducibility.

Cell

Reconstruction and Simulation of Neocortical Microcircuitry

To **simulate reconstructed microcircuits** at the level of detail described above, the **NEURON simulator was extended to run on supercomputers**

An algorithm was developed to **approximate input from the thalamus** to the central microcircuit in such a way as to satisfy experimental constraints.

Simulations exploring some of the **emergent behaviors of the reconstructed microcircuitry** reproduce a number of previous in vitro and in vivo findings and **provide insights** into the **design and functioning** of neocortical microcircuitry.

Model predictions include: a spectrum of network states ranging from synchronous to asynchronous activity; extracellular calcium regulates the network state through differential effects on synaptic dynamics; role of layers, neurons, and connection types in modulating network states;

Cell

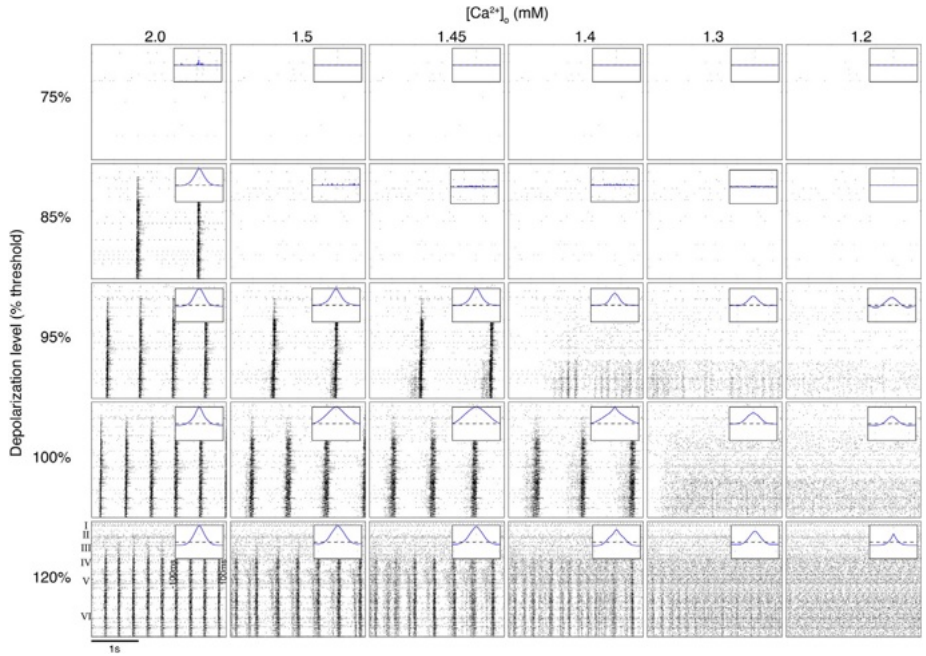
Reconstruction and Simulation of Neocortical Microcircuitry

The simulations suggest that cortical activity in vivo approaches a **critical transition** along the **synchronous asynchronous spectrum.**

Around this transition, spiking activity is highly correlated with fine temporal structure in synaptic input.

Maximal discrimination between spatially segregated inputs, the generation of fine temporal structures such as triplets, and soloist-like and chorister-like behavior all emerge close to the transition.

A recent study has experimentally characterized the plasticity mechanisms for maintaining the network close to this transition.

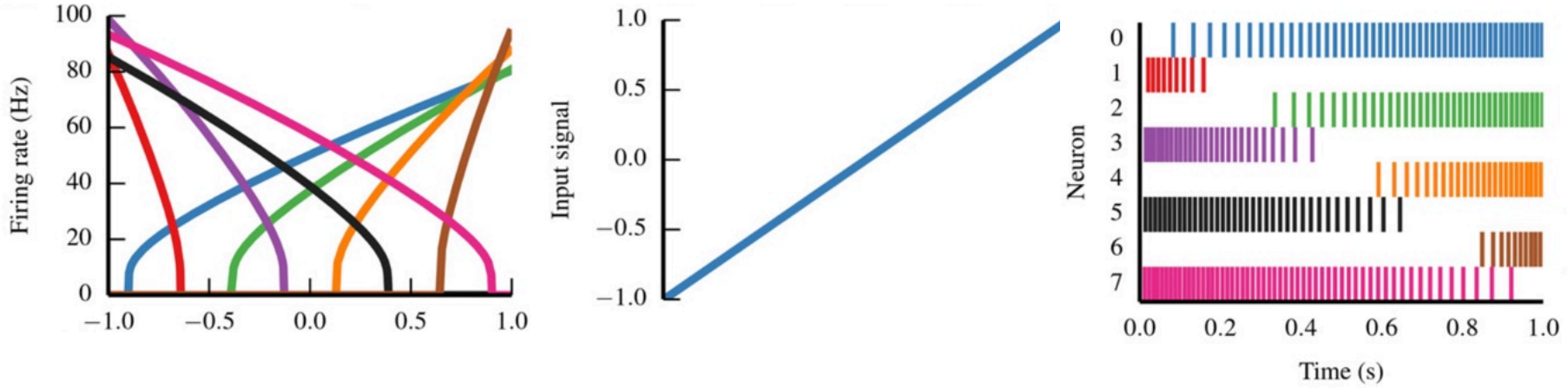


Neuro & Biomorphic Engineering LAB

The Neural Engineering Framework (NEF)

Representation

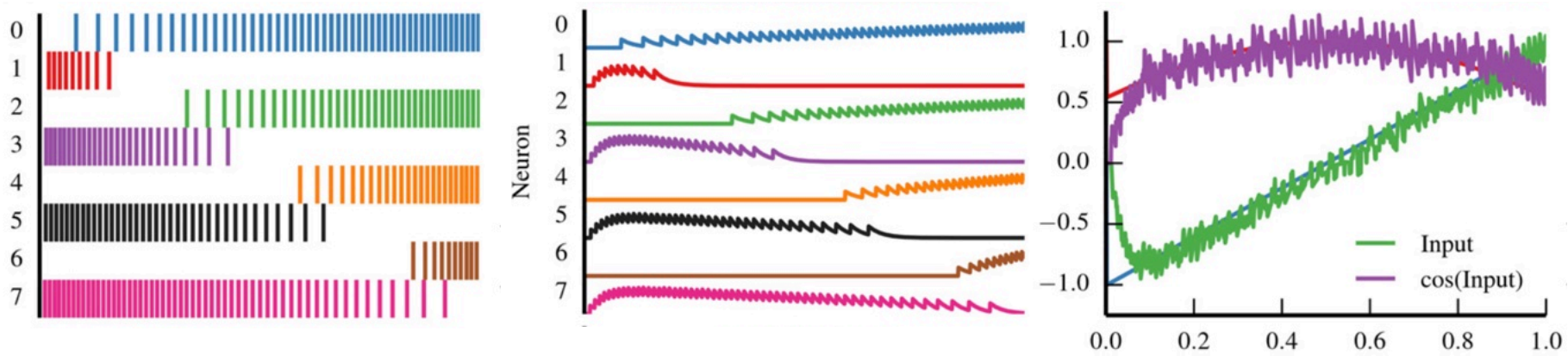
- NEF represents **neuronal information** with **time-varying vectors** of real numbers, which can by manipulated with conventional mathematics.
- In the encoding process, the input signal drives each neuron based on its **tuning curve**, which describes how likely that neuron is to respond to a given input signal (function of the ***gain*** of a neuron (how quickly the activity rises), the ***bias*** (the activity of a neuron given no signal), and the ***encoding weight***. Tuning curves can be determined for any type of neuron.



The Neural Engineering Framework (NEF)

Decoding

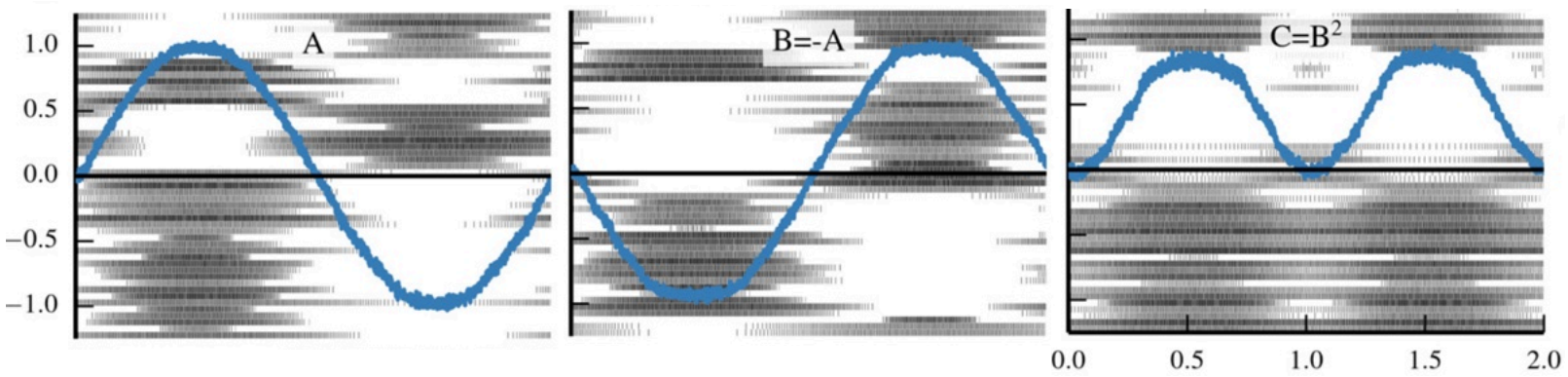
- In the decoding process, the spike trains are first **filtered** with an **exponentially decaying** filter accounting for the process of a spike generating a postsynaptic current. Those filtered spike trains are **summed together** with **weights** that are determined by solving a least-squares minimization problem.
- Decoding weights is typically performed using points sampled from the vector space that the population represents



The Neural Engineering Framework (NEF)

Transformation

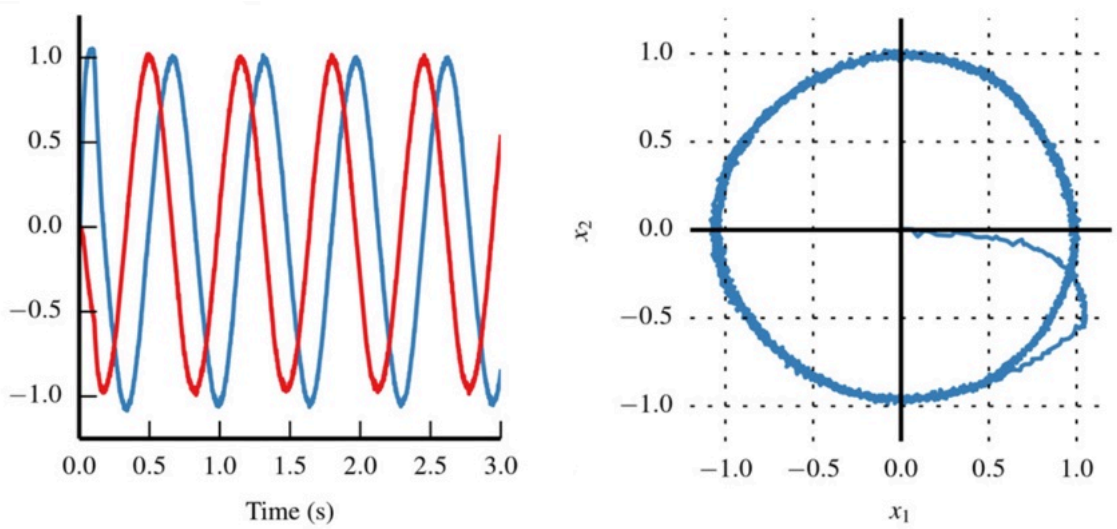
- Many factors affect synaptic strength. With NEF we summarize those factors in a scalar connection weight.
- In order to compute a function, we set the connection weights between two populations to be the product of the decoding weights for that function in the first population, the encoding weights for the downstream population, and any linear transform.



The Neural Engineering Framework (NEF)

Dynamics

- While feedforward vector transformations suffice to describe some neural systems, many require persistent activity through recurrent connections.
- When recurrent connections are introduced, the vectors represented by neural populations can be thought of as state variables in a dynamical system. The equations governing dynamics in such a system can be designed and analyzed using the methods of control theory, and translated into neural circuitry using the principles of representation and transformation.



Nengo

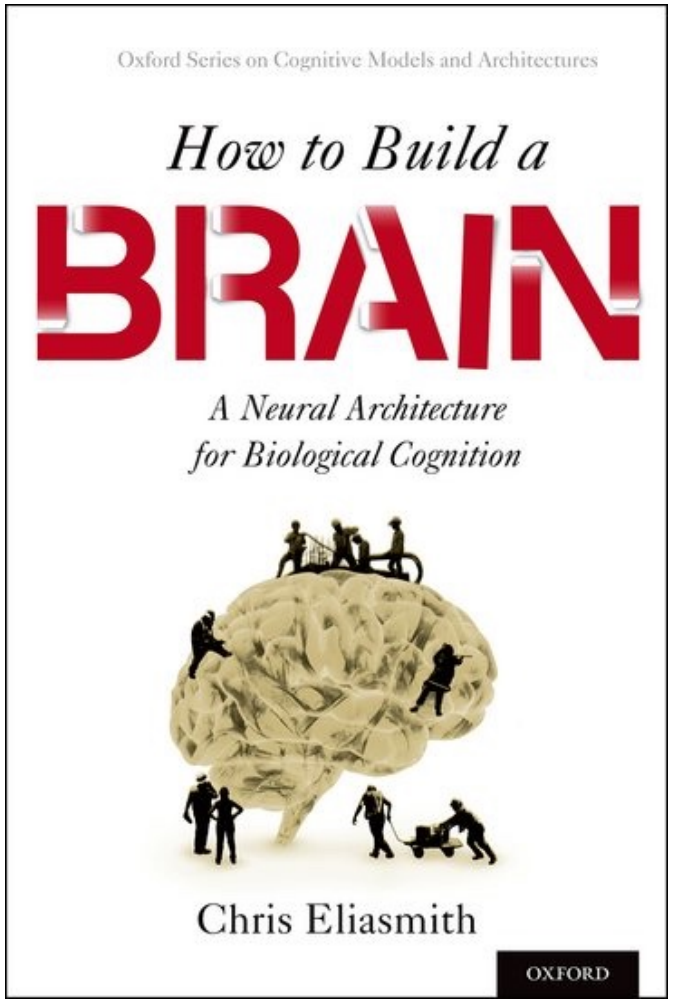
- A software package for simulating large-scale neural systems.
- To use Nengo, you define groups of neurons in terms of what they represent, and then form connections between neural groups in terms of what computation should be performed on those representations.
- It uses the Neural Engineering Framework (NEF) to solve for the appropriate synaptic connection weights to achieve this desired computation.
- Supports various kinds of learning and make detailed spiking neuron models that implement complex high-level cognitive algorithms.
- Has been used to implement motor control, visual attention, serial recall, action selection, working memory, attractor networks, inductive reasoning, path integration, and planning.

Nengo

- Nengo 2.0 was recently implemented in Python. You should install Python from: https://www.python.org/downloads/. Code was tested with Python's interpreter version 3.5.2

- Great chose for a python IDE is PyCharm Community: https://www.jetbrains.com/pycharm/download/ Windows version is preferred (stability)

- Installation instruction for Nengo: https://www.pythonhosted.org/nengo/getting_started.html



The Neural Engineering Framework (NEF)

Representation

Representation Formalism

- Value being represented: x
- Neural activity: a
- Neuron index: i

Encoding and decoding

- Have to define both to define a code
- Lossless code (e.g. Morse Code):
  - Encoding: $a = f(x)$
  - Decoding: $x = f^{-1}(a)$
- Lossy code:
  - Encoding: $a = f(x)$
  - Decoding: $x \sim g(a)$

NOEL

The Neural Engineering Framework (NEF)

Representation

Distributed representation

- Not just one neuron per *x* value but rather many different *a* values for a single *x*
  - Encoding: $a_i = f_i(x)$
  - Decoding: $x \sim g(a_0, a_1, a_2, a_3, \ldots)$

- *Linear decoding:*

$$\hat{x} = \sum_i a_i d_i$$

  - *Elegant and simple*
  - *Works fine with non-linear encoding (!)*

NOEL

The Neural Engineering Framework (NEF)

Representation

Example: binary representation

- Encoding:

$$a_i = \begin{cases} 1 & \text{if } x \mod 2^i > 2^{i-1} \\ 0 & \text{otherwise} \end{cases}$$

- Decoding:

$$\hat{x} = \sum_i a_i 2^{i-1}$$

- Suppose: x = 13
- Encoding: $a_1 = 1$, $a_2 = 0$, $a_3 = 1$, $a_4 = 1$
- Decoding: x ~ 1 * 1+ 0 * 2 + 1 * 4 + 1 * 8

NOEL

The Neural Engineering Framework (NEF)

Representation

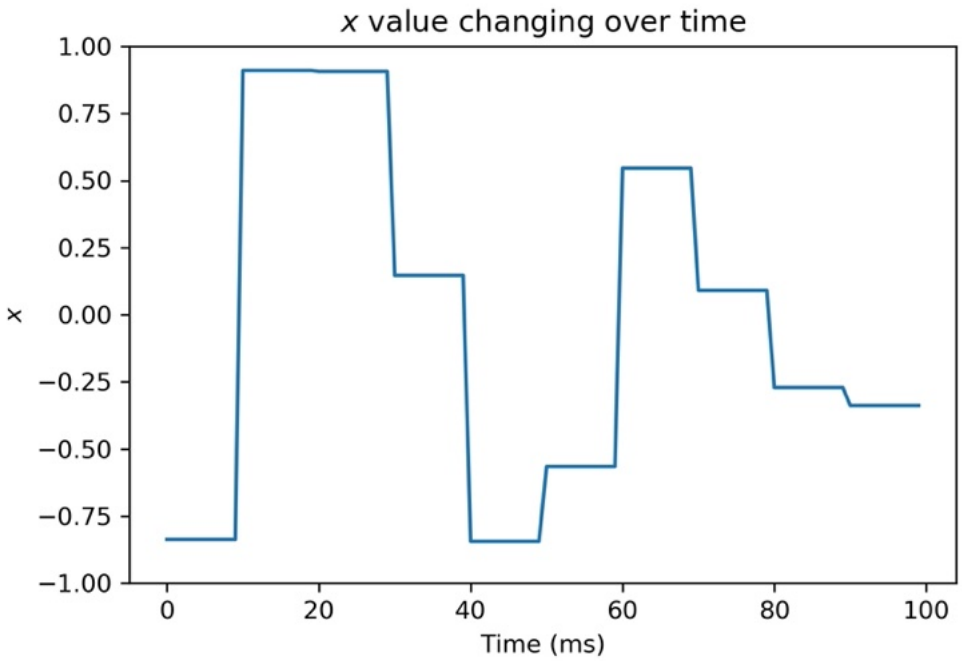
Neuron encoding

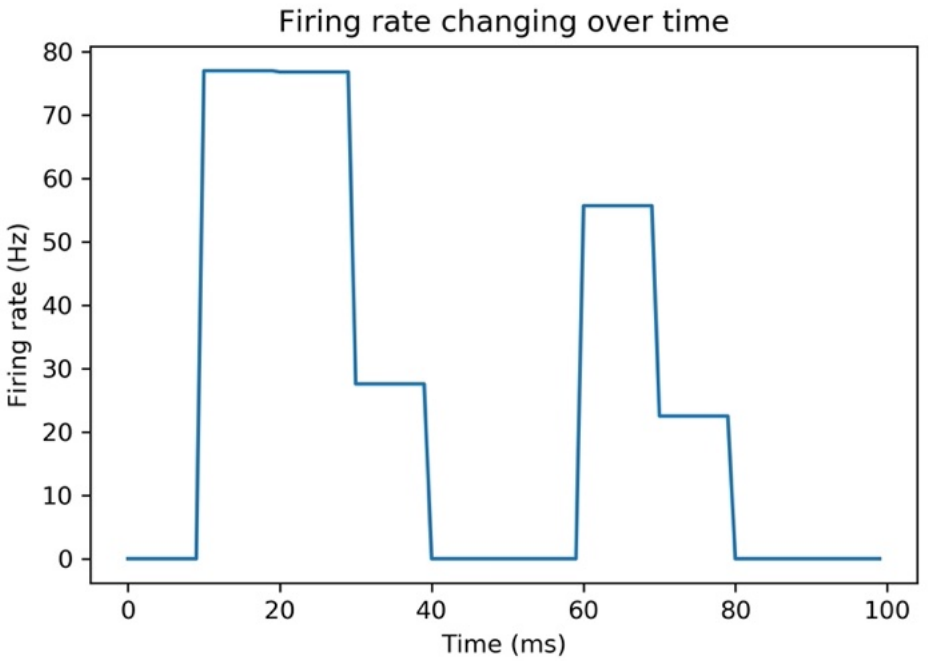
- Encoding: $a_i = f_i(x)$
- Firing rate goes up as total input current J goes up: $a_i = G_i(J(x))$
- What is G_i?
  - Depends on how detailed a neuron model we want

The Neural Engineering Framework (NEF)

Temporal Representation

- Unfortunately, with a naive rate code, the formula for $G[J(t)]$ is an *average* firing rate over a long period of time (what does it mean to be firing at 10Hz for 10ms?)
- We need to model what's actually happening to that neuron over time



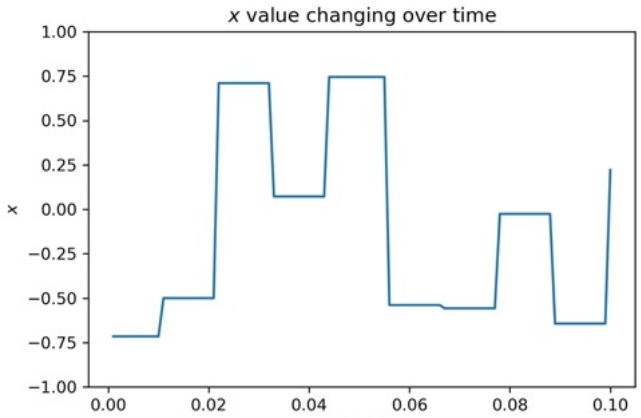


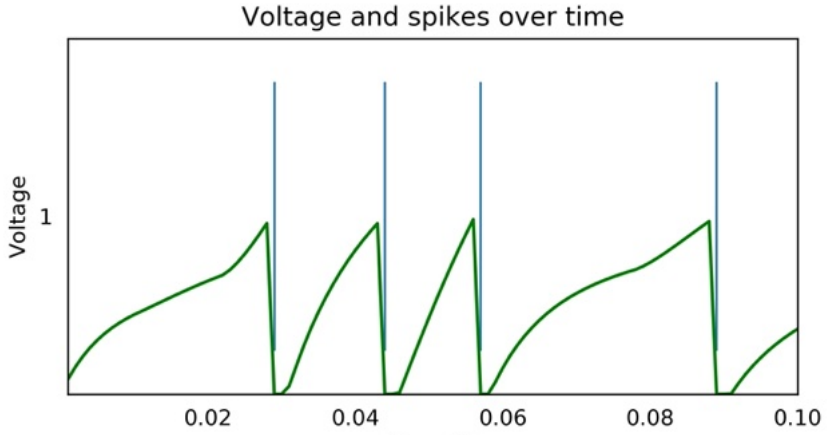
The Neural Engineering Framework (NEF)

Representation

From the Integrate and Fire Model…

- Time per spike equals time to threshold + refractory period
- From f = 1/t (frequency - time relationship): $a = \frac{1}{t_{th} + t^{ref}}$
- From the LIF model: $u(t) = u_{rest} + R \cdot I_0 \left[1 - e^{-\frac{t - t_0}{\tau}}\right]$
- By solving for t_{th}, we get: $a = \frac{1}{t^{ref} - \tau \ln(1 - \frac{V_{th}}{R \cdot I_0})}$

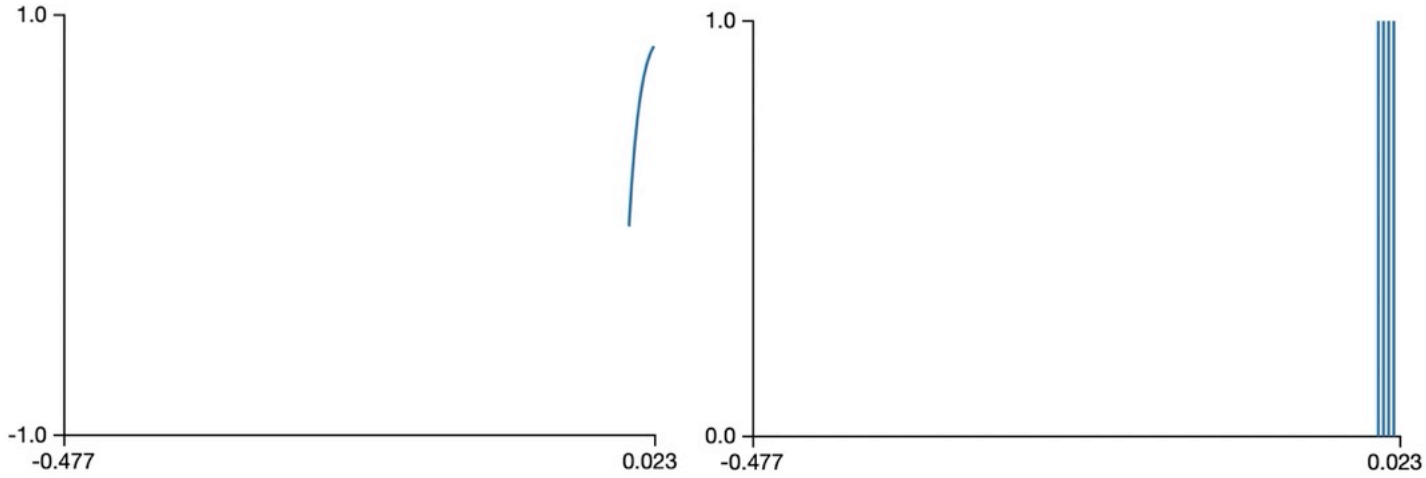




The Neural Engineering Framework (NEF)

Representation

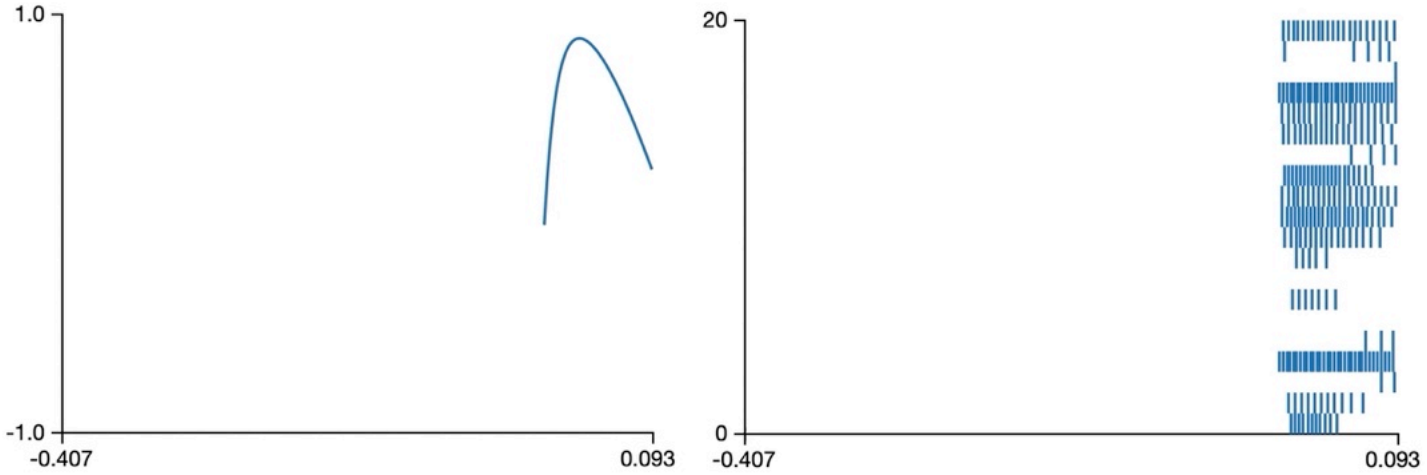
From the Integrate and Fire Model…



The Neural Engineering Framework (NEF)

Representation

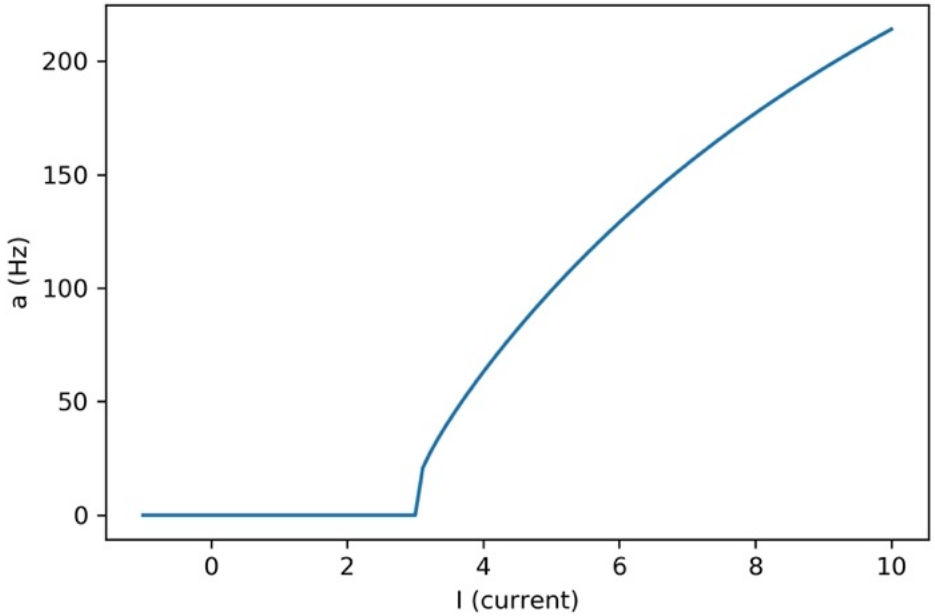
From the Integrate and Fire Model…



The Neural Engineering Framework (NEF)

Representation

- When RC = 0.02 and refractory is 0.002 the a-I curve will look like:

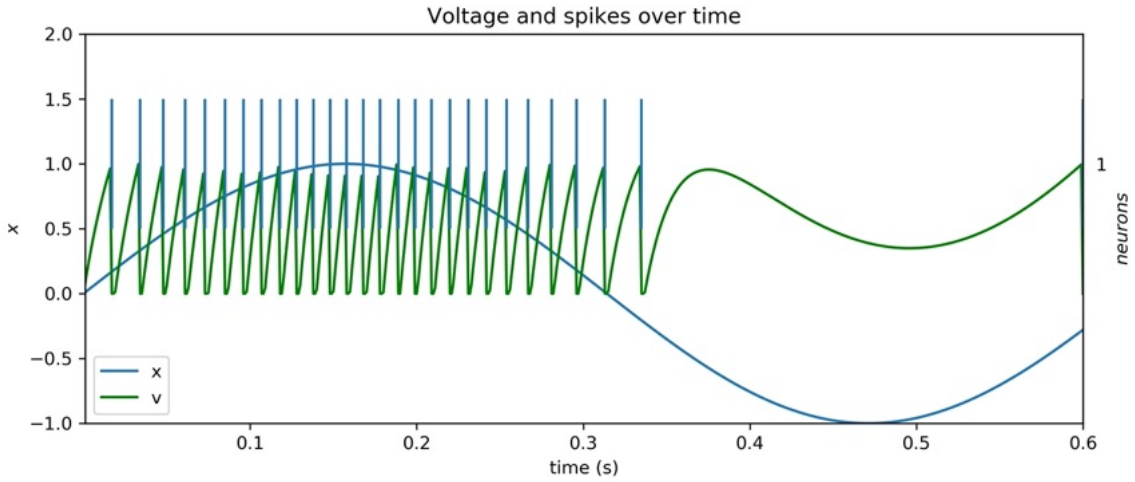


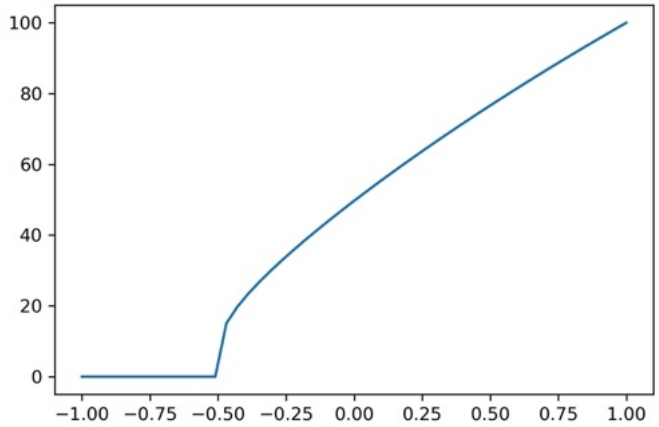
- These are called "response functions"
- How much neural firing changes with change in current
- Similar for many classes of cells (e.g. pyramidal cells - most of cortex)
- This is the G_i function in the NEF: it can be pretty much anything

The Neural Engineering Framework (NEF)

Representation

- For example (a sinus wave):





The Neural Engineering Framework (NEF)

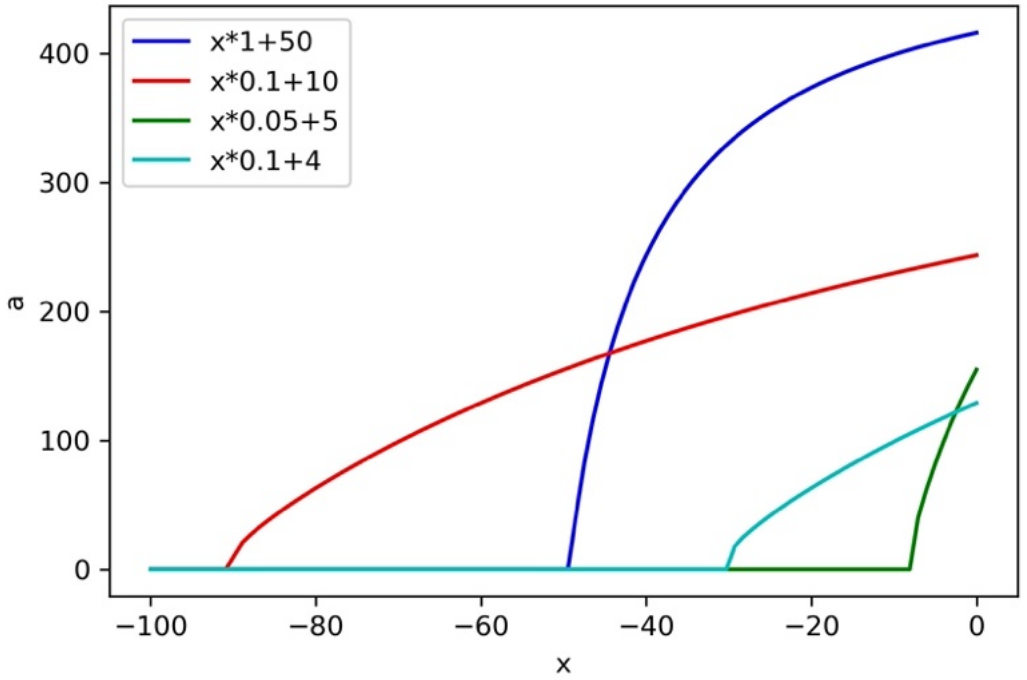
Representation

- Weaknesses:
  - These are "point" neurons (i.e. we're assuming the body of the neuron is just one infinitessimal point, rather than having lots and lots of compartments, each with its own RC circuit)
  - Assumes R is a constant
  - Assumes V_{th} is a constant
  - Assumes no adaptation (it's harder for a neuron to fire after it has recently fired)
  - Doesn't consider nonlinearities at the input

The Neural Engineering Framework (NEF)

Representation

- Neurons seem to be sensitive to particular values of x (receptive fields, visual stimulation…)
- How are neurons 'tuned' to a representation? or…
- What's the mapping between *x* and *a*?
- Encoding: $a_i = G_i(J(x))$
- One simple mapping might be: $J = \alpha x + J^{bias}$



The Neural Engineering Framework (NEF)

Representation

- There's usually some *x* which gives a maximum firing rate ...and thus a maximum *J…*
- Firing rate (and J)) decrease as you get farther from the preferred x value
- So something like: $J = \alpha[sim(x, x_{pref})] + J^{bias}$
- What sort of similarity measure?

The Neural Engineering Framework (NEF)

Representation

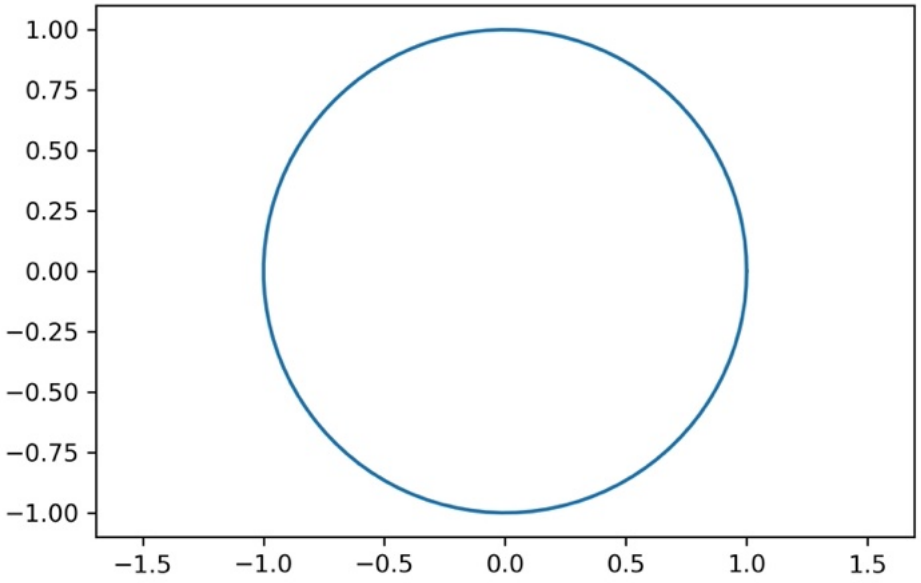
- A general form:

$$a_i = G_i[\alpha_i x \cdot e_i + J_i^{bias}]$$

A positive gain term

the encoder, or the preferred direction vector with a unit length

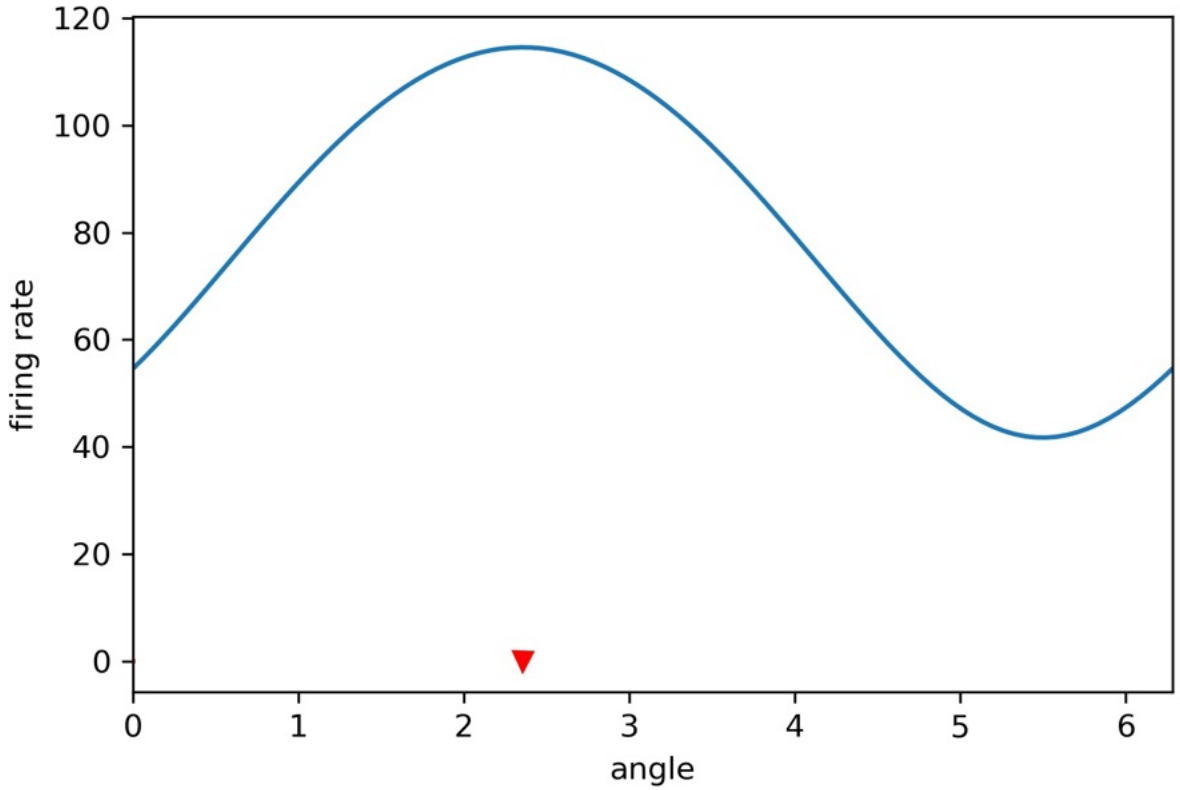
- In 1D e is either +1 or -1, in 2D:



The Neural Engineering Framework (NEF)

Representation

- If the preferred direction is: [-1, 1] (indicated in red), the response curve would be:



The Neural Engineering Framework (NEF)

Representation

- Encoding: $a_i = G_i(J(x))$
- For decoding we want: $\hat{x} = \sum_i a_i d_i$
- But where do we get d_i from?
- By minimising the average error over all x: $E = \frac{1}{2} \int_{-1}^{1} (x - \hat{x})^2 \, dx$
- Where: $E = \frac{1}{2} \int_{-1}^{1} \left(x - \sum_{i}^{N} a_i d_i \right)^2 dx$
- Minimising the derivative with respect to d_i:

$$\frac{\partial E}{\partial d_i} = \frac{1}{2} \int_{-1}^{1} 2 \left[x - \sum_j a_j d_j \right] (-a_i) \, dx$$

$$\frac{\partial E}{\partial d_i} = - \int_{-1}^{1} a_i x \, dx + \int_{-1}^{1} \sum_j a_j d_j a_i \, dx$$

NOEL

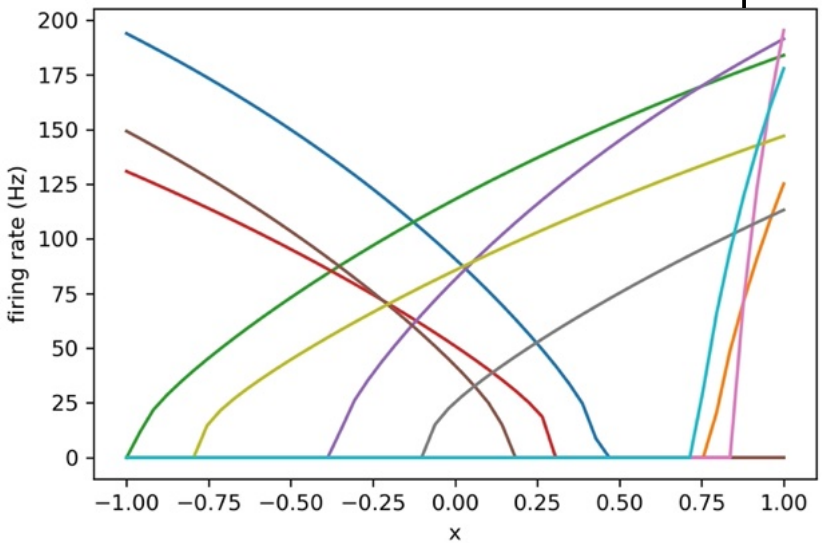
The Neural Engineering Framework (NEF)

Representation

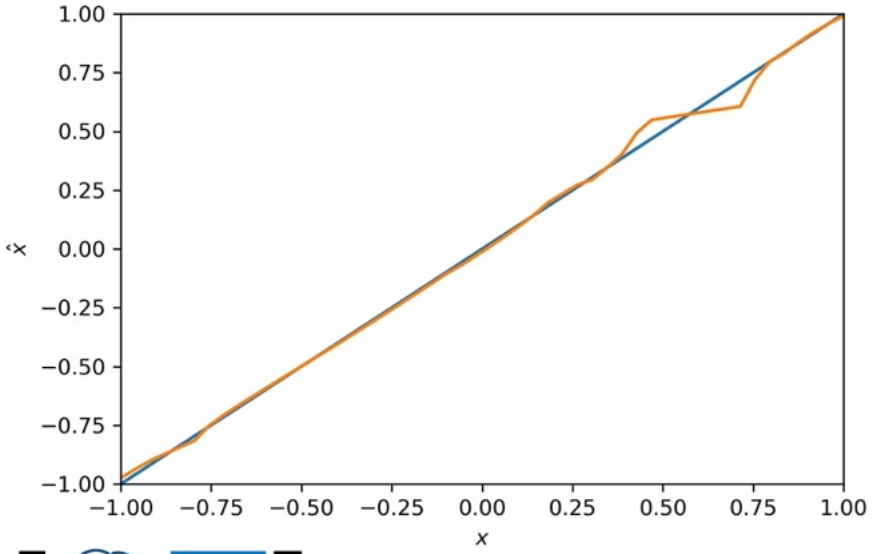
- Setting: $\frac{\partial E}{\partial d_i} = 0$
- Getting us to: $$\int_{-1}^{1} a_i x \, dx = \int_{-1}^{1} \sum_j (a_j d_j a_i) \, dx$$
$$\int_{-1}^{1} a_i x \, dx = \sum_j \left(\int_{-1}^{1} a_i a_j \, dx \right) d_j$$
- That's a system of N equations and N unknowns
- Approximate the integral by sampling over x: $$\sum_x a_i x/S = \sum_j \left(\sum_x a_i a_j/S \right) d_j$$
S is the number of x values to use (S for samples) $\Upsilon = \Gamma d$
- Solving the linear system: $d = \Gamma^{-1} \Upsilon$
$$d_i = \sum_j \Gamma^{-1}_{ij} \Upsilon_j$$

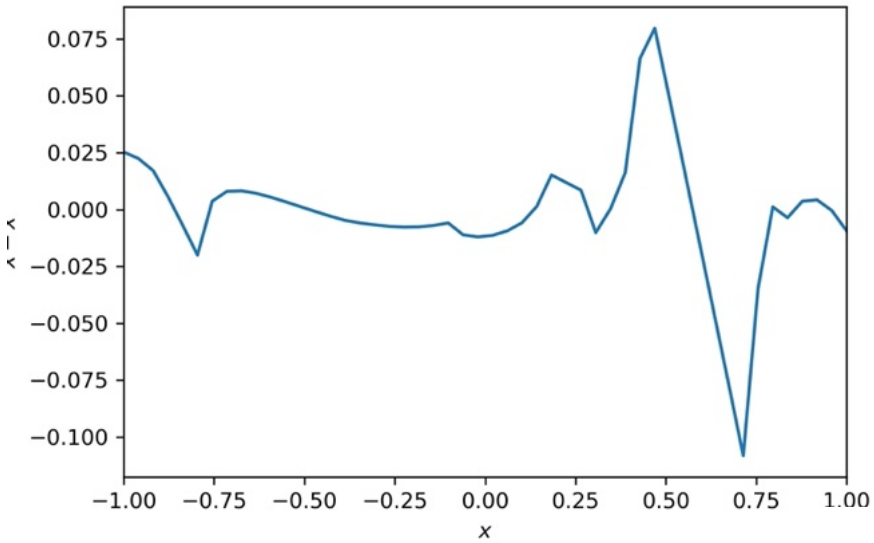
The Neural Engineering Framework (NEF)

Representation



Defaults to LIF neurons, with random gains and biases for neurons between 100-200hz over -1,1



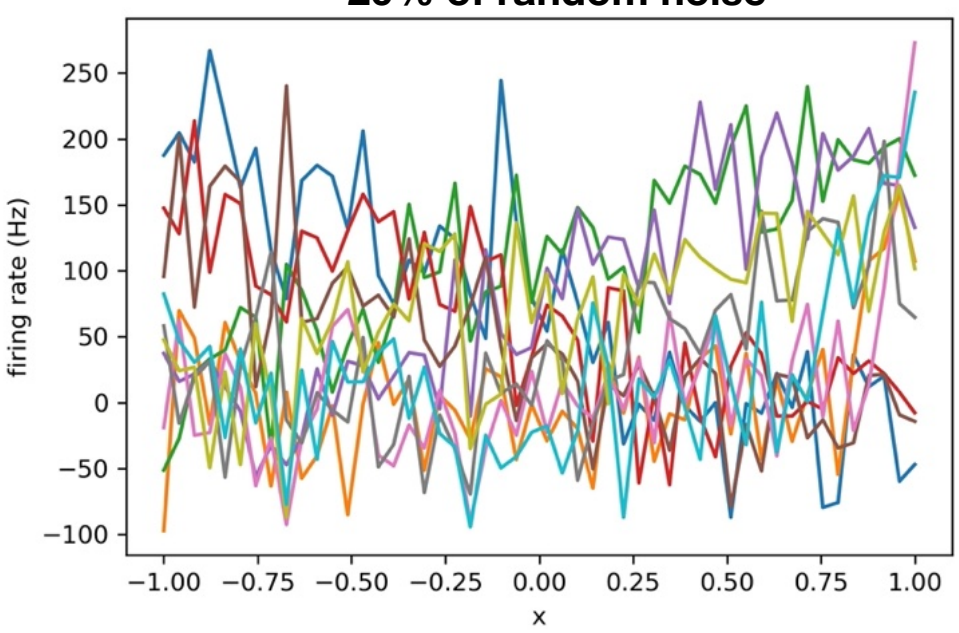


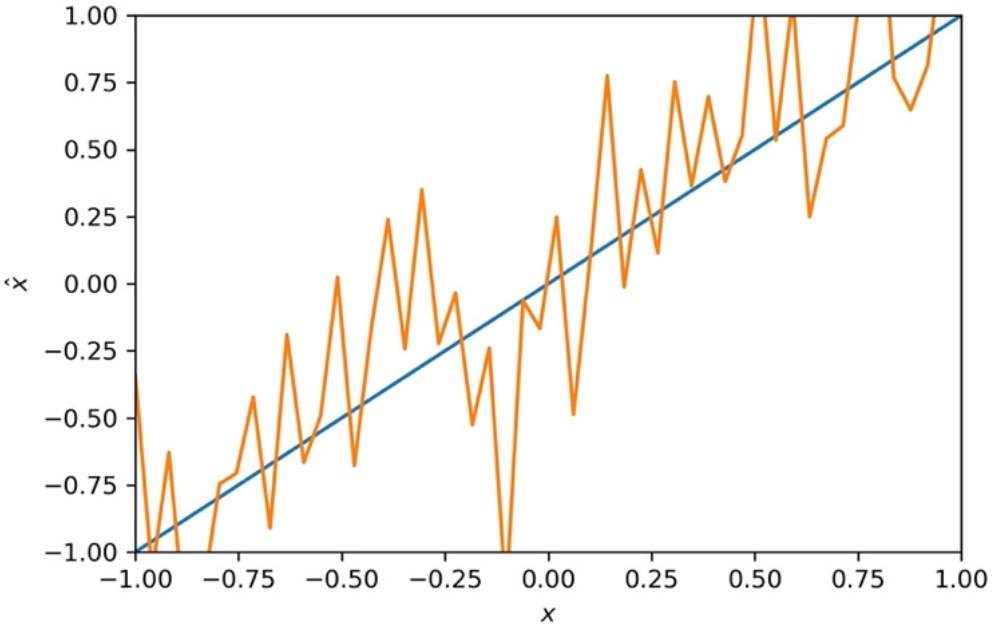
The Neural Engineering Framework (NEF)

Representation

- Neurons are not perfect: they are a noisy entities
  see: https://icwww.epfl.ch/~gerstner/SPNM/node33.html

20% of random noise





The Neural Engineering Framework (NEF)

Representation

- Handling noise is one of the more efficient aspects in neuromorphic systems. Unlike engineered representation, biological representation explicitly address the effect of noise
- Taking noise into account (Introduce noise term η)

$$\hat{x} = \sum_i (a_i + \eta) d_i$$

$$E = \frac{1}{2} \int_{-1}^{1} (x - \hat{x})^2 \, dx d\eta$$

$$= \frac{1}{2} \int_{-1}^{1} \left(x - \sum_i (a_i + \eta) d_i \right)^2 dx d\eta$$

$$= \frac{1}{2} \int_{-1}^{1} \left(x - \sum_i a_i d_i - \sum \eta d_i \right)^2 dx d\eta$$

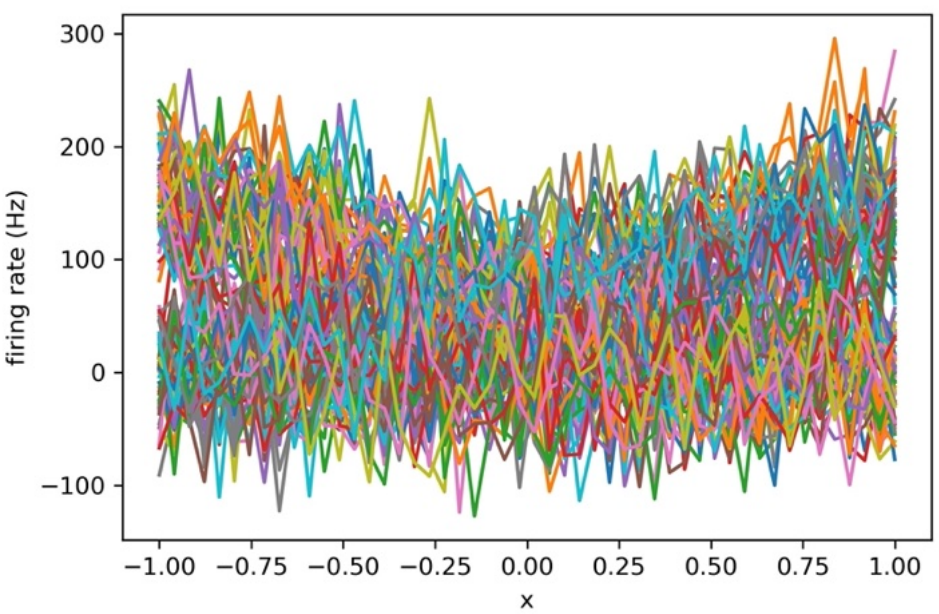
- Assume noise is gaussian, independent, mean zero, and has the same variance for each neuron

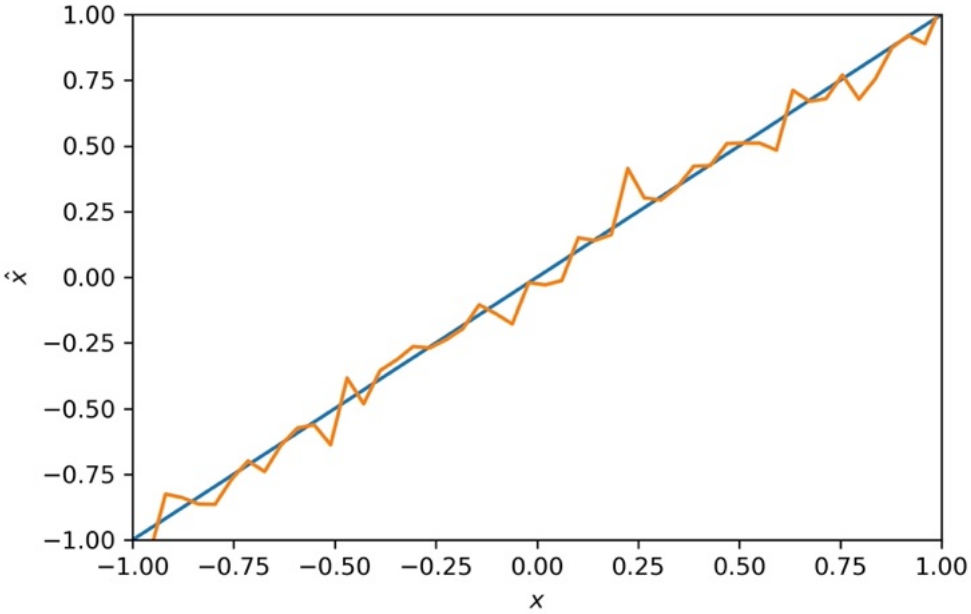
$$E = \frac{1}{2} \int_{-1}^{1} \left(x - \sum_i a_i d_i \right)^2 dx + \sigma^2 \sum_i d_i^2$$

The Neural Engineering Framework (NEF)

Representation

20% of random noise





The Neural Engineering Framework (NEF)

Representation

$$E = \frac{1}{2}\int_{-1}^{1}\left(x - \sum_i a_i d_i\right)^2 dx + \sigma^2 \sum_i d_i^2$$

Error due to static distortion (i.e. the error introduced by the decoders themselves)

Error due to noise

- Noise error is proportional to $1/N$
- Distortion error is proportional to $1/N^2$
- When the number of neurons is greater than 100, the error is dominated by the noise term

The Neural Engineering Framework (NEF)

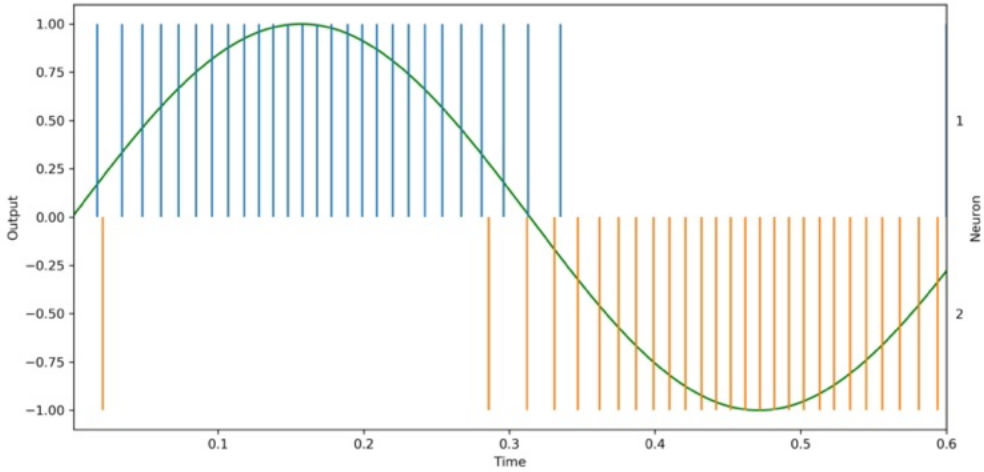
Temporal Representation

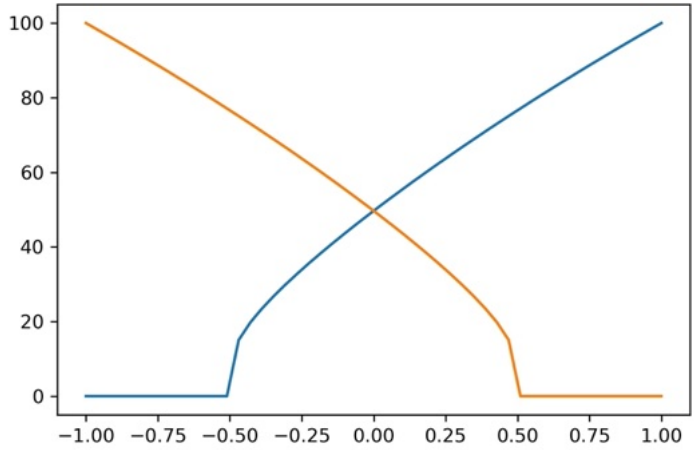
- We covered representation of a vector space by a population of neurons.
- We converted a vector x into neural activity values a_i, and then converted those back to the original value x
- What happens if x changes over time?
  - Maybe: $a = G[\alpha e \cdot x + J^{bias}]$, so...
    $a(t) = G[\alpha e \cdot x(t) + J^{bias}]$
    where $G[J(t)] = \frac{1}{\tau_{ref} - \tau_{RC} ln(1 - \frac{1}{J(t)})}$
- How do decode this? Can we stick with linear decoding?

The Neural Engineering Framework (NEF)

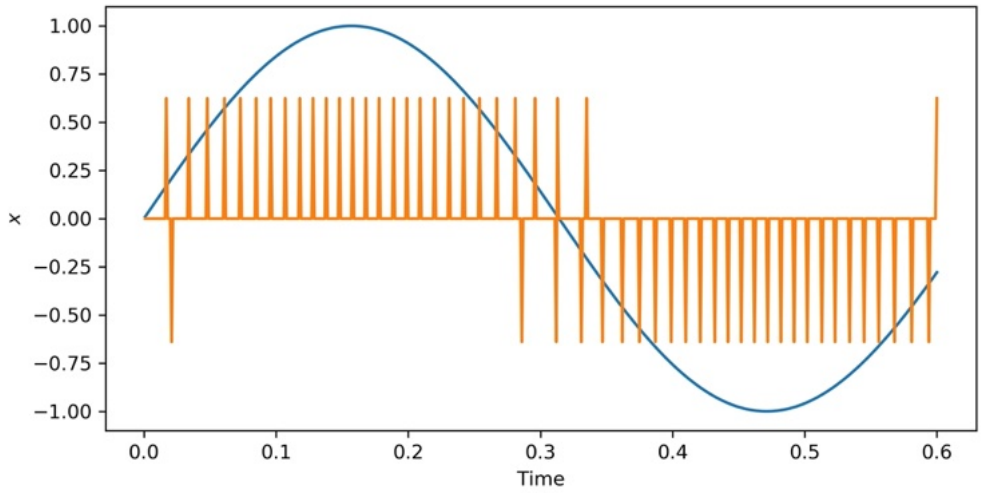
Temporal Representation

- Consider:





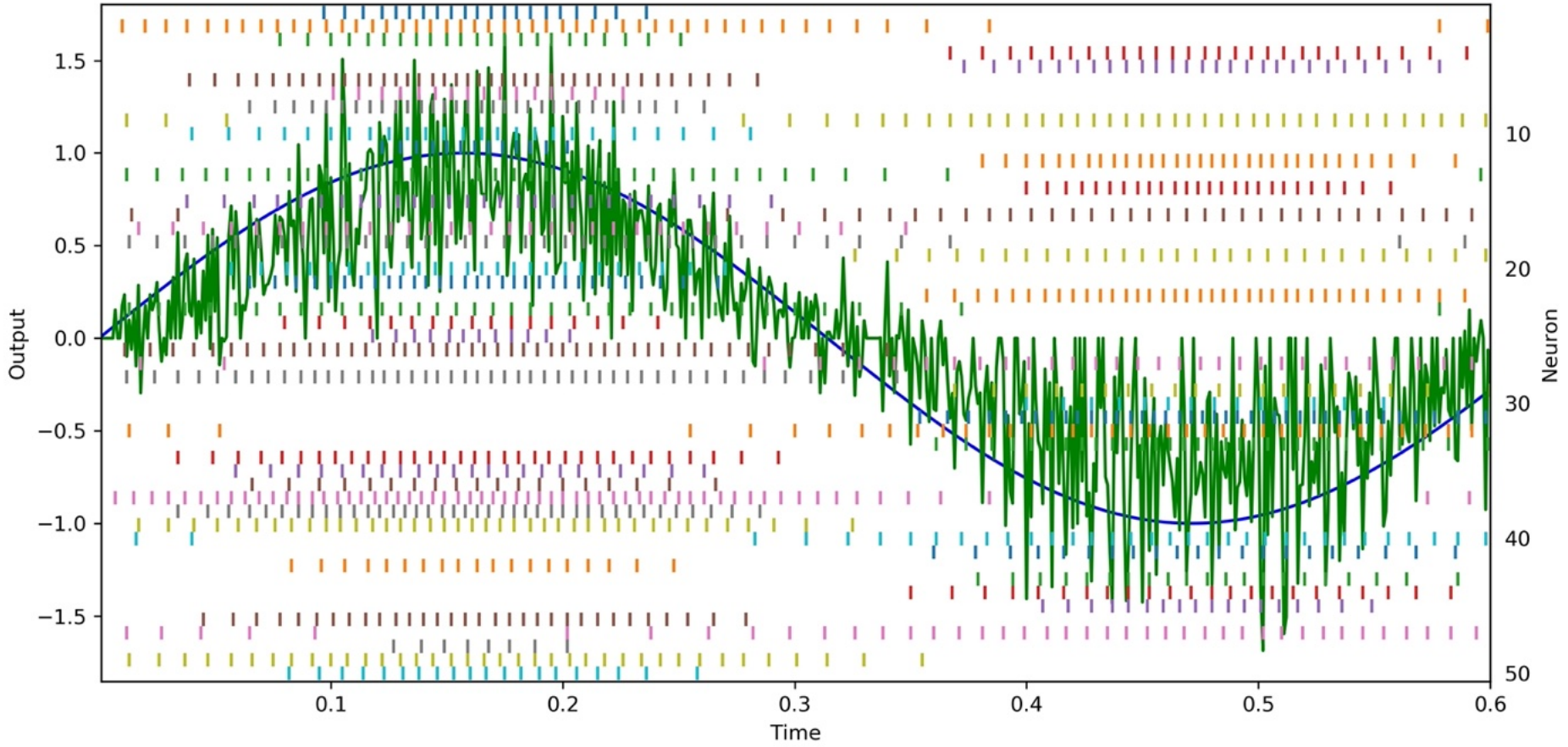
- Decoding x using a sum of activities with $\hat{x}(t) = \sum_i a_i(t) d_i$:



The Neural Engineering Framework (NEF)

Temporal Representation

- If we add more neurons:



The Neural Engineering Framework (NEF)

Temporal Representation

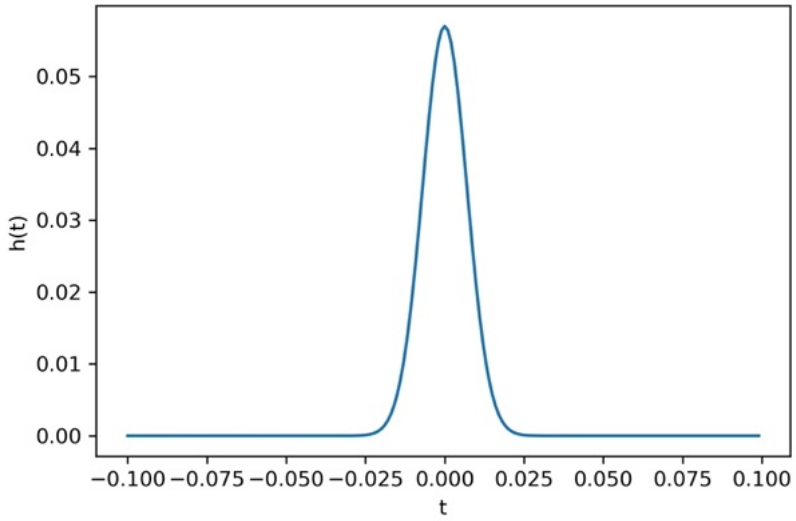
- What's happening here?
  - At a given instant in time t, only a very small number of neurons are firing.
  - If no neurons are firing at that instant, $x = 0$
  - There are usually only one or two neurons firing at any given time
  - As dt gets smaller, the problem gets even worse
  - What can we do?
    - We want a spike to contribute to other points in time
    - So we want to convert a train of spikes into some continuous function

The Neural Engineering Framework (NEF)

Temporal Representation

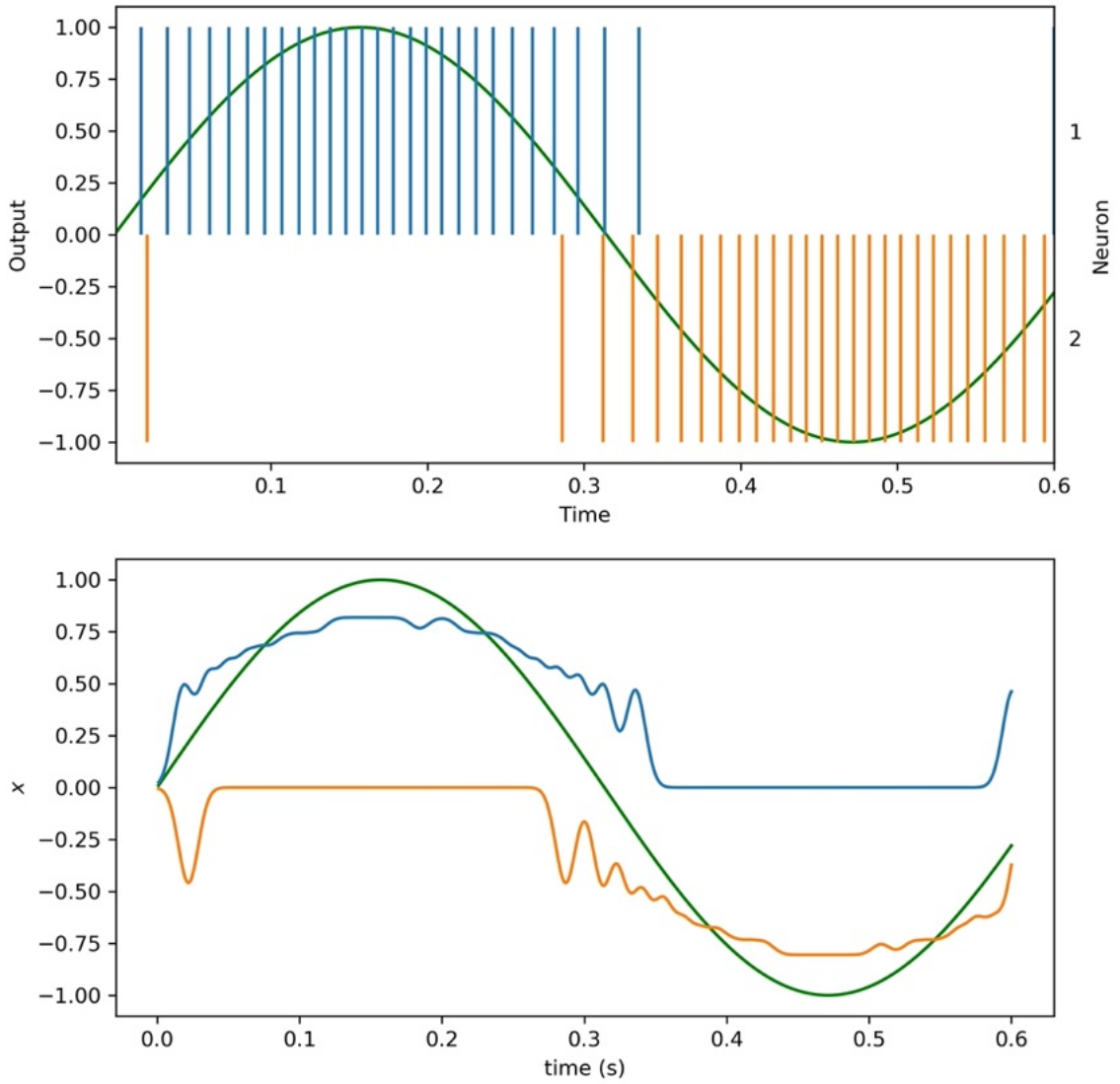
- Consider convolution: $\hat{x}(t) = \sum_i a_i(t) * h(t)d_i$

  where h(t) is a gaussian: $h(t) = e^{-t^2/2\sigma^2}$



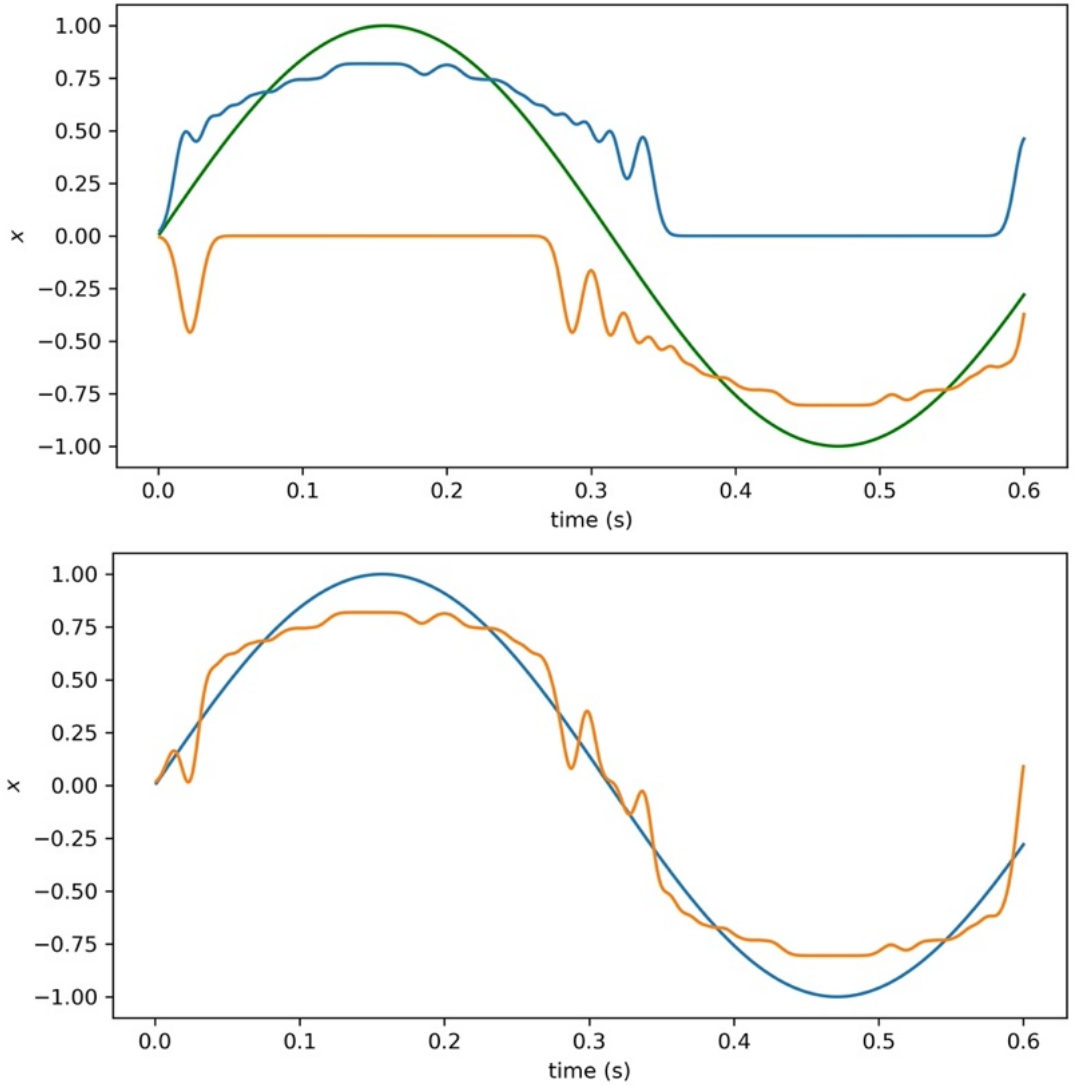
The Neural Engineering Framework (NEF)

Temporal Representation



The Neural Engineering Framework (NEF)

Temporal Representation



The Neural Engineering Framework (NEF)

Temporal Representation

- Can we do better?
- Consider the two neurons case, where $d_1 = -d_2$:

$$\hat{x}(t) = \sum_i a_i(t) * h(t)d_i$$
$$\hat{x}(t) = a_1(t) * h(t)d_1 + a_2(t) * h(t)d_2$$
$$\hat{x}(t) = (a_1(t) - a_2(t)) * h(t)d_1$$
$$\hat{x}(t) = (a_1(t) - a_2(t)) * h(t)$$
$$\hat{x}(t) = r(t) * h(t)$$

- The convolution theorem: convolution turns into multiplication when you do a Fourier transform (https://en.wikipedia.org/wiki/Convolution_theorem):

$$\hat{X}(\omega) = R(\omega)H(\omega)$$
$$E(\omega) = (X(\omega) - R(\omega)H(\omega))^2$$

- Now we can take the derivative and set it equal to zero to do the minimization:

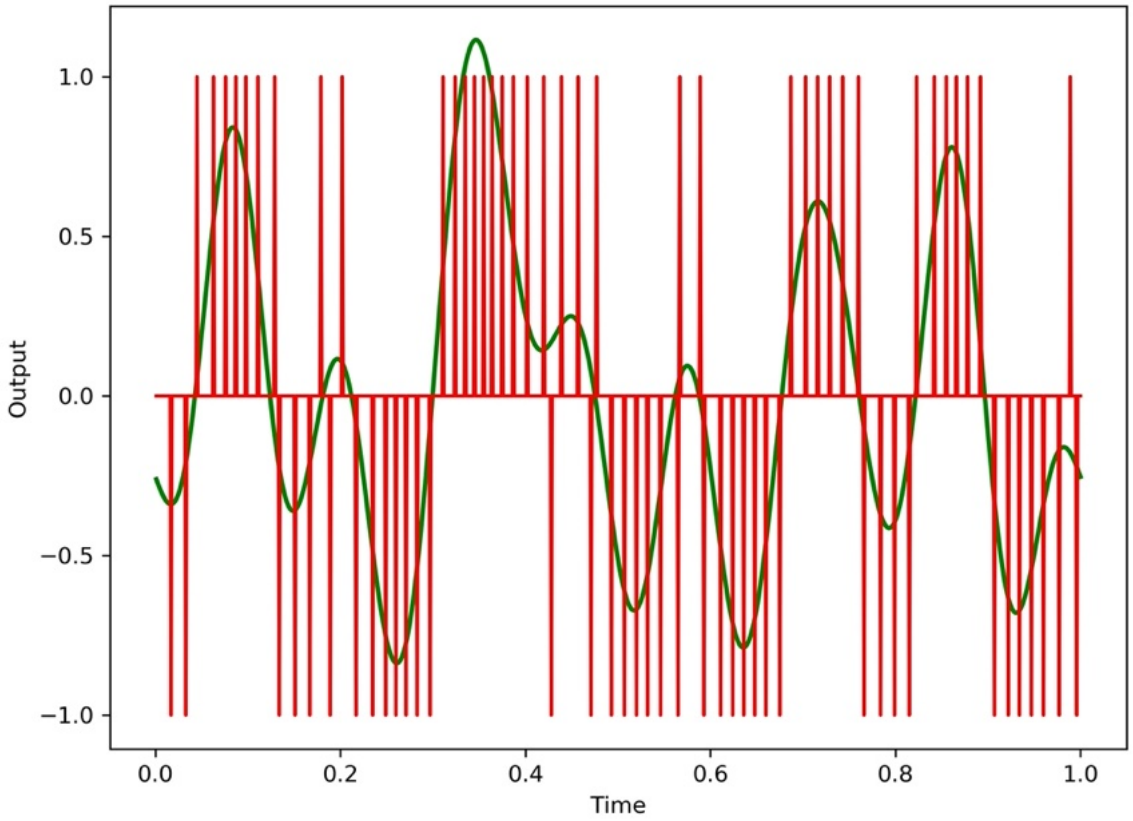
$$H(\omega) = \frac{X(\omega)R^*(\omega)}{|R(\omega)|^2}$$

The Neural Engineering Framework (NEF)

Temporal Representation

- So now we can find $H(\omega)$ given the Fourier transform of our signal $X(\omega)$ and the Fourier transform of the spiking response $R(\omega)$:

Signal and spiking responce

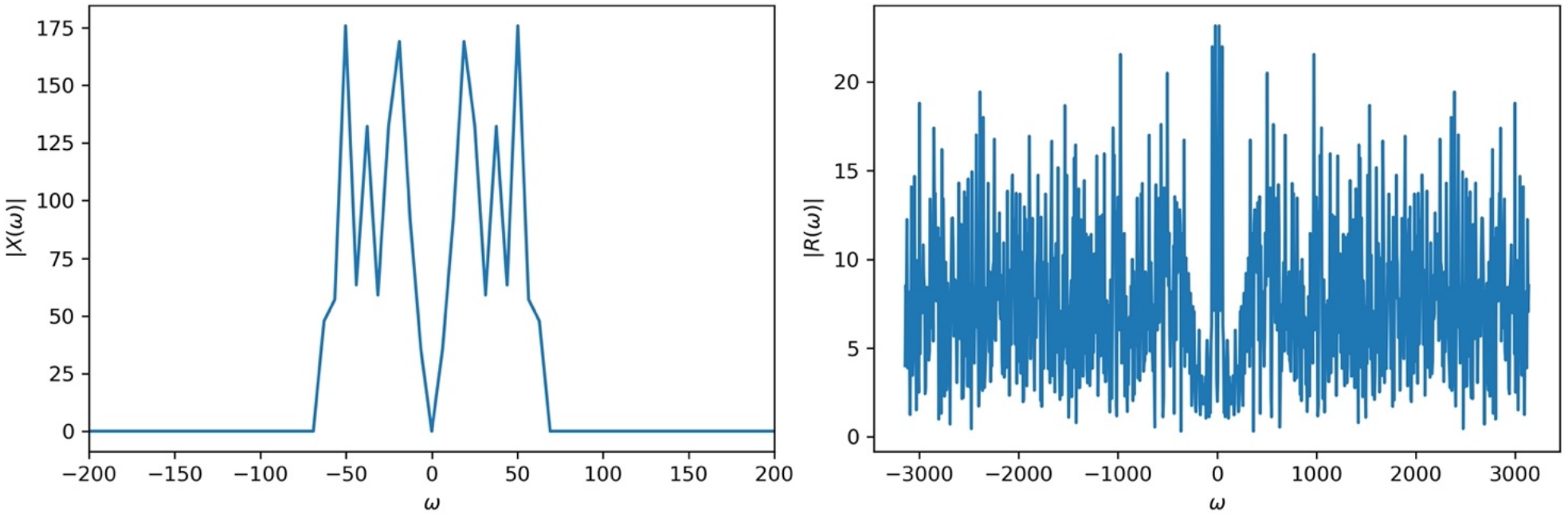


The Neural Engineering Framework (NEF)

Temporal Representation

- So now we can find $H(\omega)$ given the Fourier transform of our signal $X(\omega)$ and the Fourier transform of the spiking response $R(\omega)$:

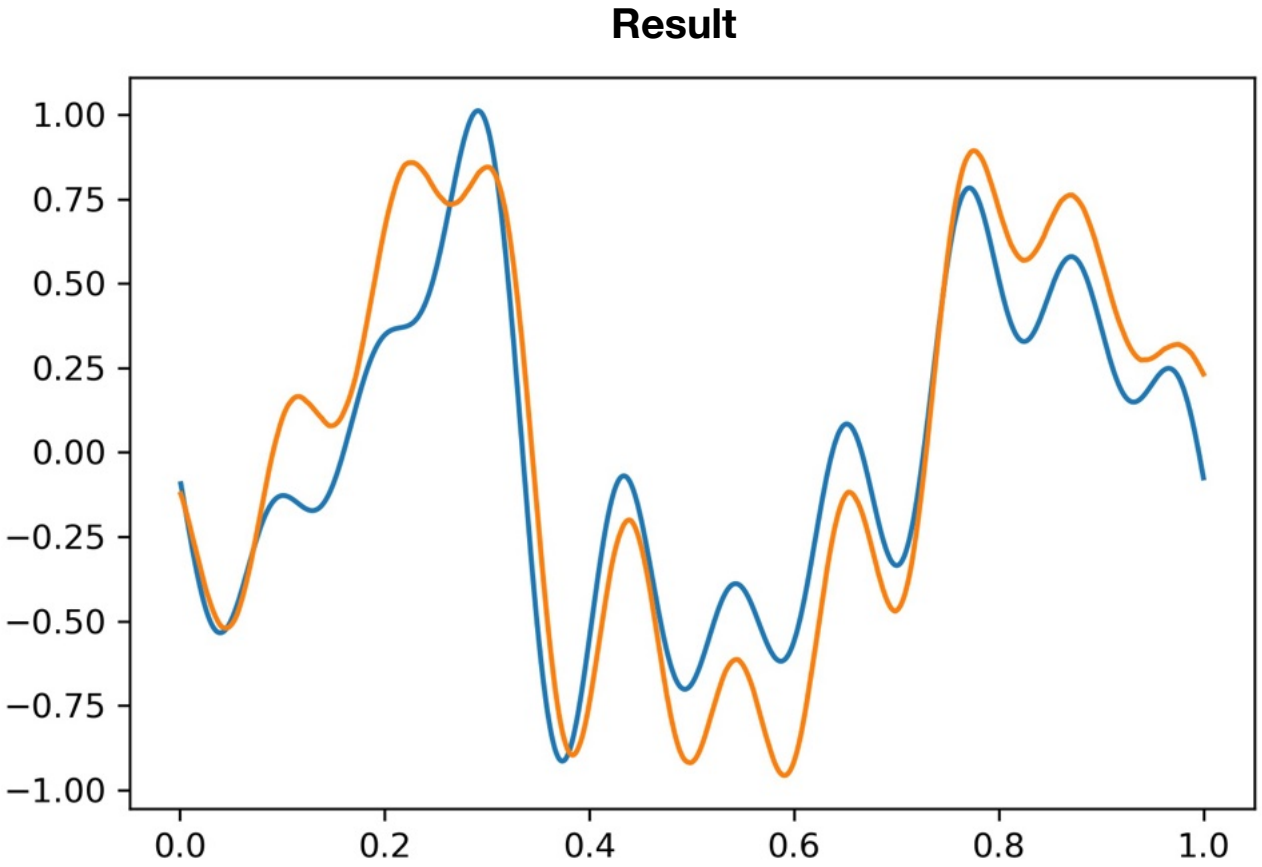
Fourier transform of signal and spiking



The Neural Engineering Framework (NEF)

Temporal Representation

- Can we generalise it?
- Same filter on a different input:



The Neural Engineering Framework (NEF)

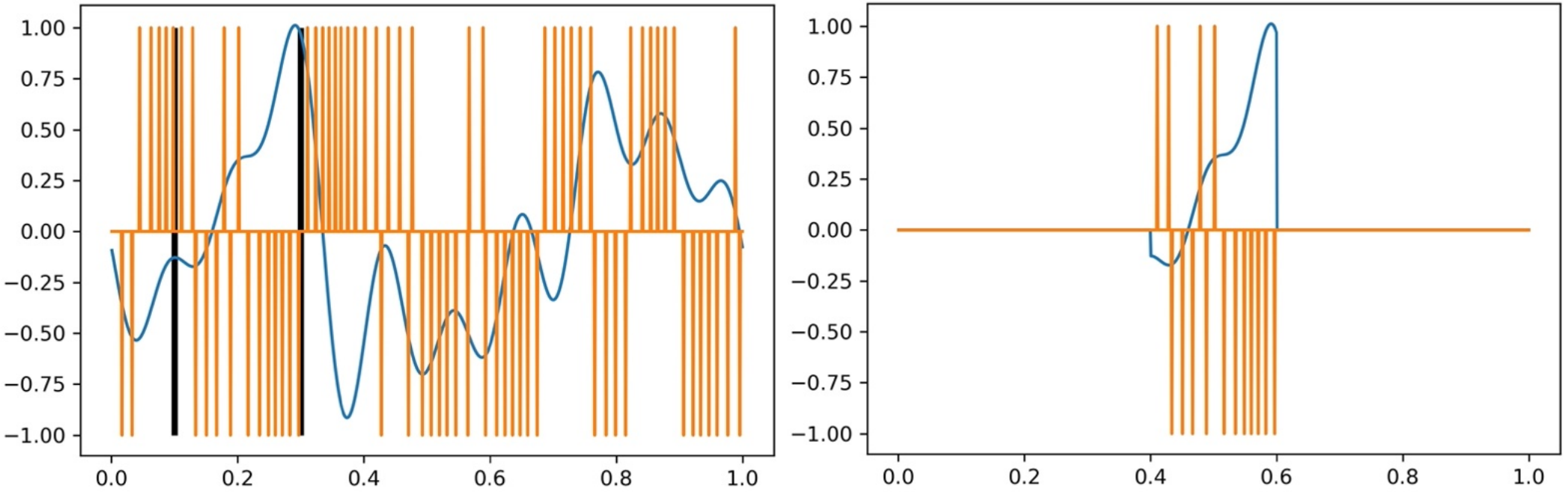
Temporal Representation

- We're only training on a pretty small set of data
  - So let's increase τ
  - This can help, but can get computationally expensive
- Another approach
  - We've already created a pretty long signal ($\mathrm{T}=1.0$ seconds)
  - We know the filter should probably decay towards zero
    - A spike shouldn't affect values far away from when it happens
  - So let's take our one existing signal and chop it up into lots of little signals

The Neural Engineering Framework (NEF)

Temporal Representation

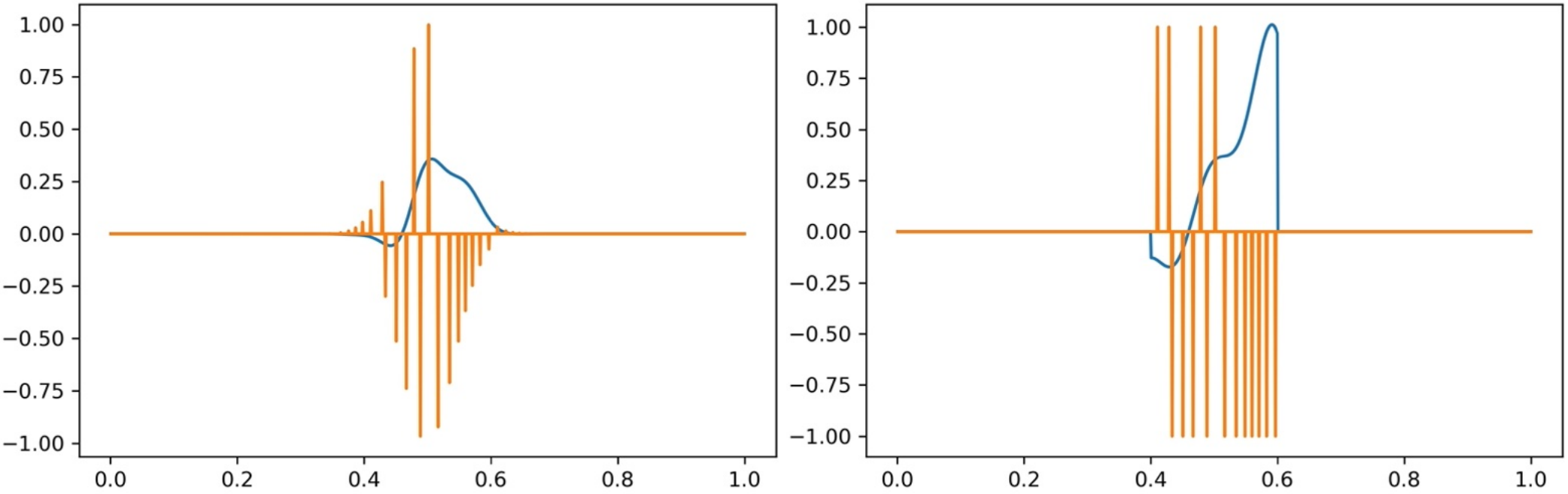
- So let's take our one existing signal and chop it up into lots of little signals



The Neural Engineering Framework (NEF)

Temporal Representation

- This introduces some edge effects at the beginning and ending of the window
- So we can do some sort of smoothing of the data (we use a Gaussian)



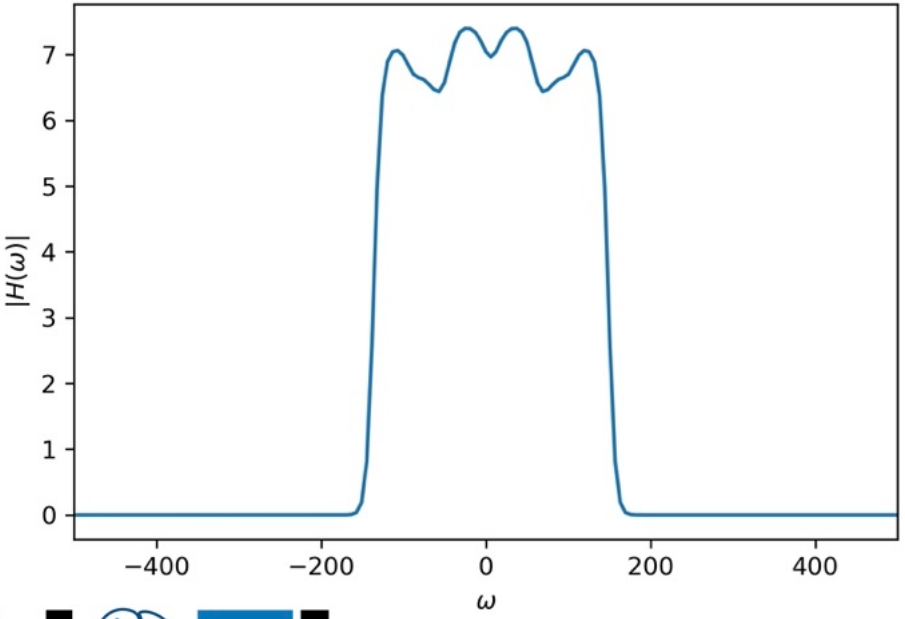
The Neural Engineering Framework (NEF)

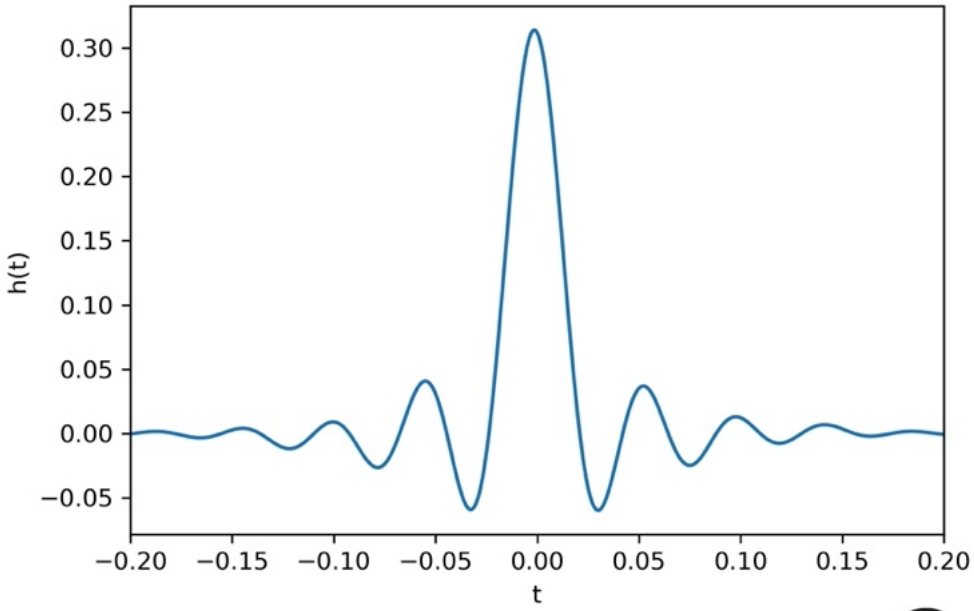
Temporal Representation

- Last graph can be thought of as one trial, which we perform at every Δt and then average over:

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

- So we're windowing and averaging in time doing convolution in frequency (it's kind of unusual to see convolution in frequency), creating = an efficient method for estimation and getting:

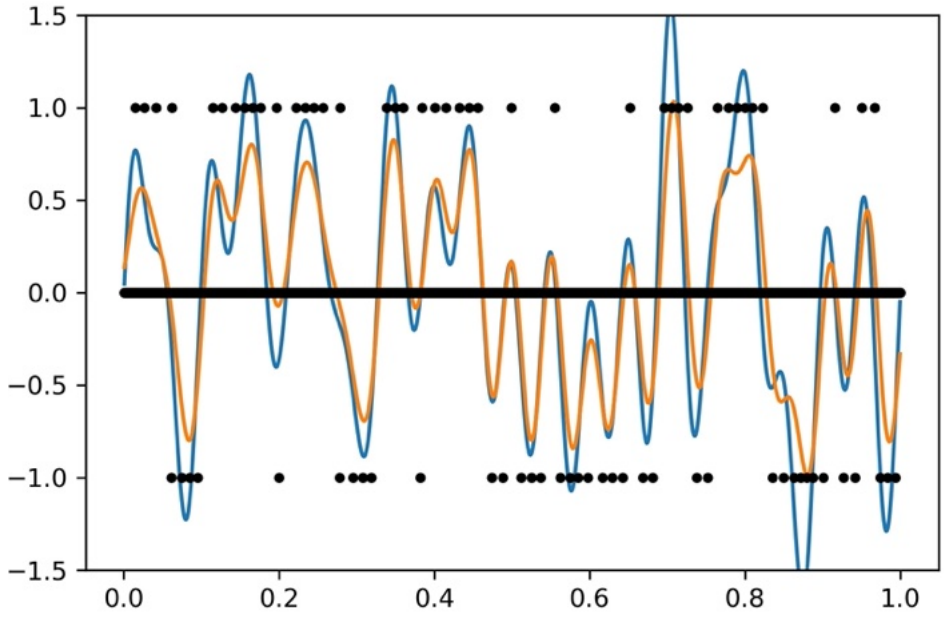


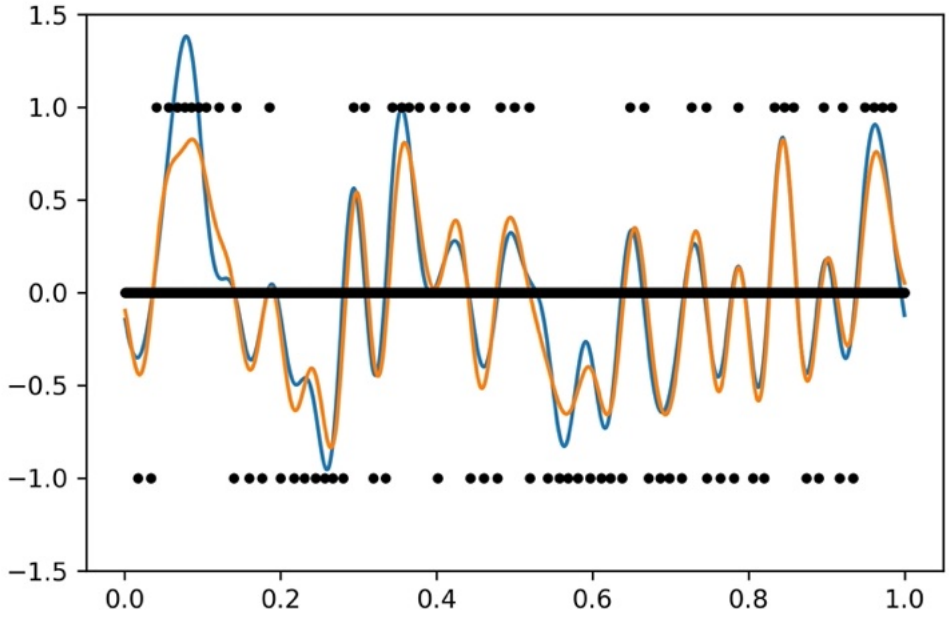


The Neural Engineering Framework (NEF)

Temporal Representation

- Results:



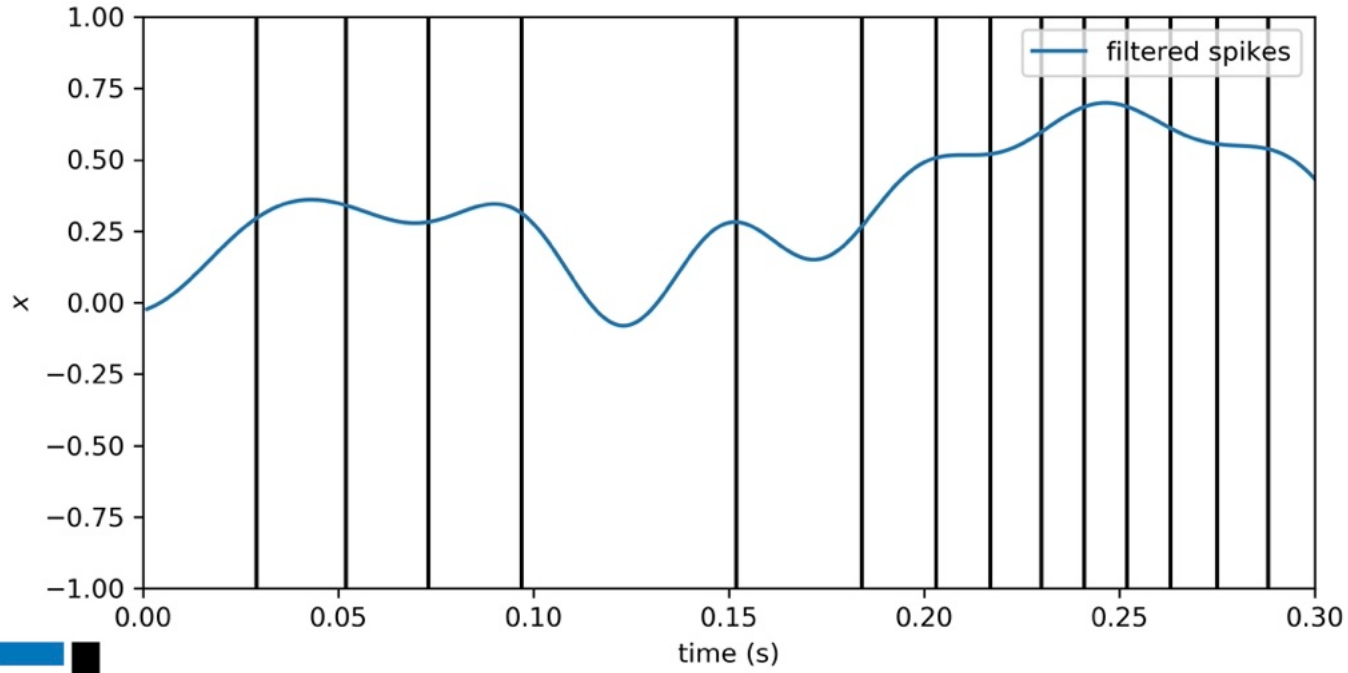


The Neural Engineering Framework (NEF)

Temporal Representation

NOTE

- The decoder works better if it is made for the right range of frequencies
- Here, we've only done this for the case of 2 neurons, but we could use the same filter for many more neurons
- Using our filter, every time a spike occurs, we practically replace the spike with the filter shape shape. For example:



The Neural Engineering Framework (NEF)

Temporal Representation

- If we're analyzing the data after the fact, this is fine
- But what if we're just sitting here observing the neural activity and wanting to know what's being represented right now?
- Notice that the blue line (the filtered value we're using for decoding) starts going up *before* the spike happens
- The filtered value is using information ***from the future***.
- This means that when we look at x this way, we're getting a better estimate of x than it is possible for other neurons to get (assuming neurons aren't psychic)
- What could we do instead?

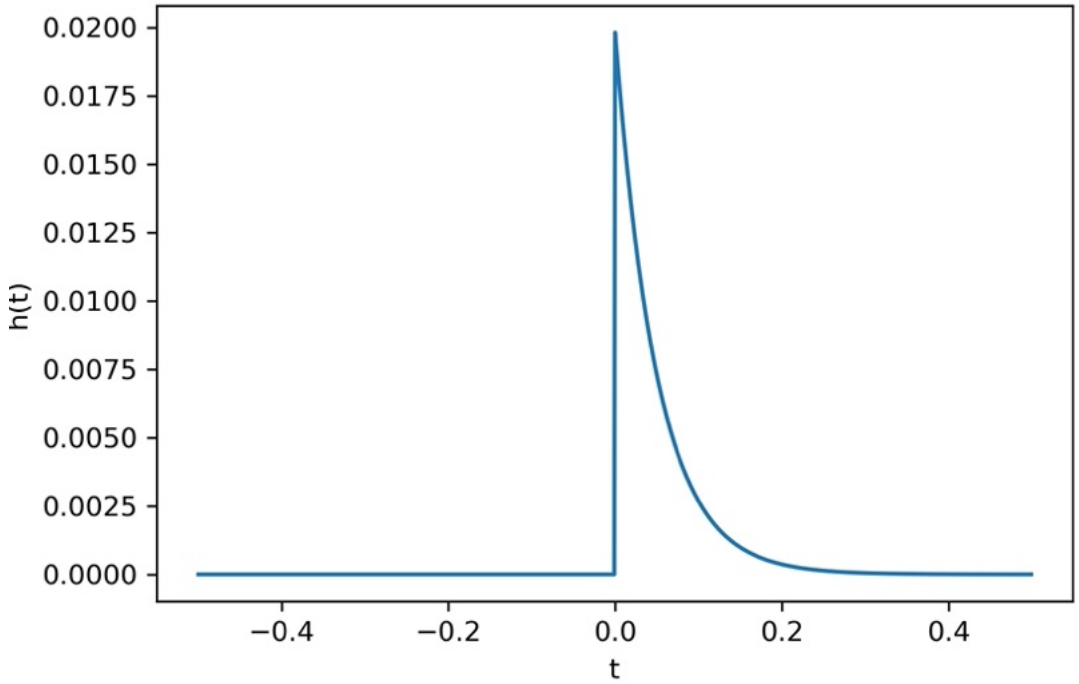
The Neural Engineering Framework (NEF)

Temporal Representation

- A biologically plausible model:

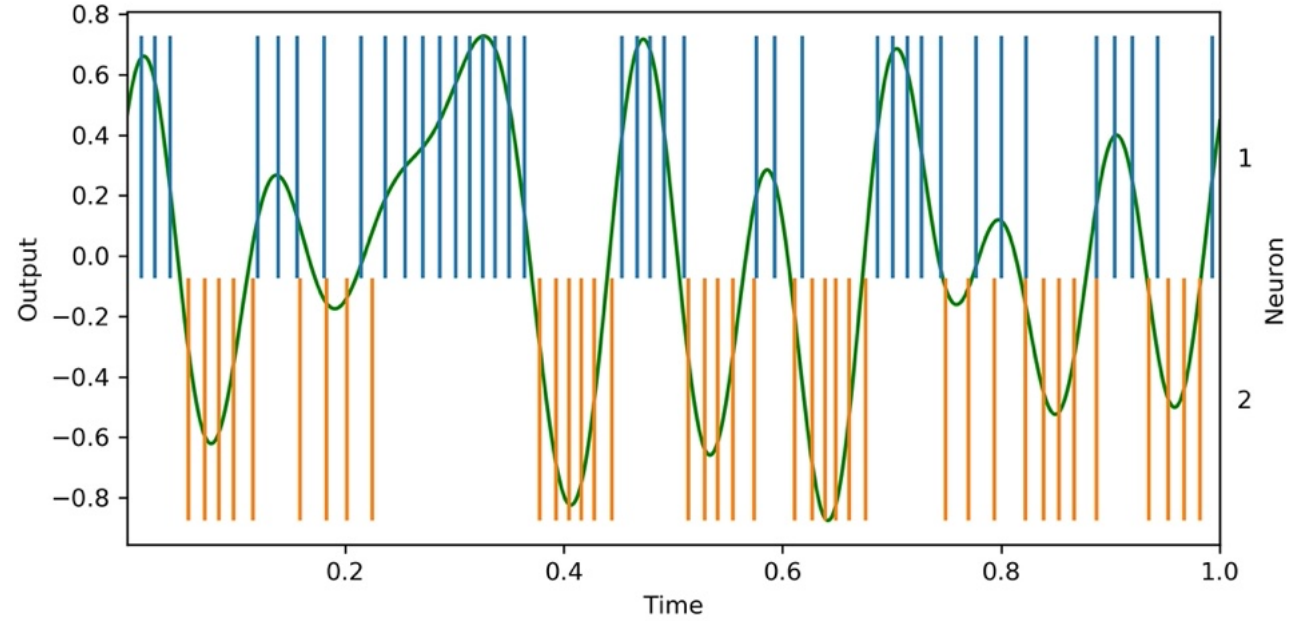
$$h(t) = \begin{cases} e^{-t/\tau} & \text{if } t > 0 \\ 0 & \text{otherwise} \end{cases}$$

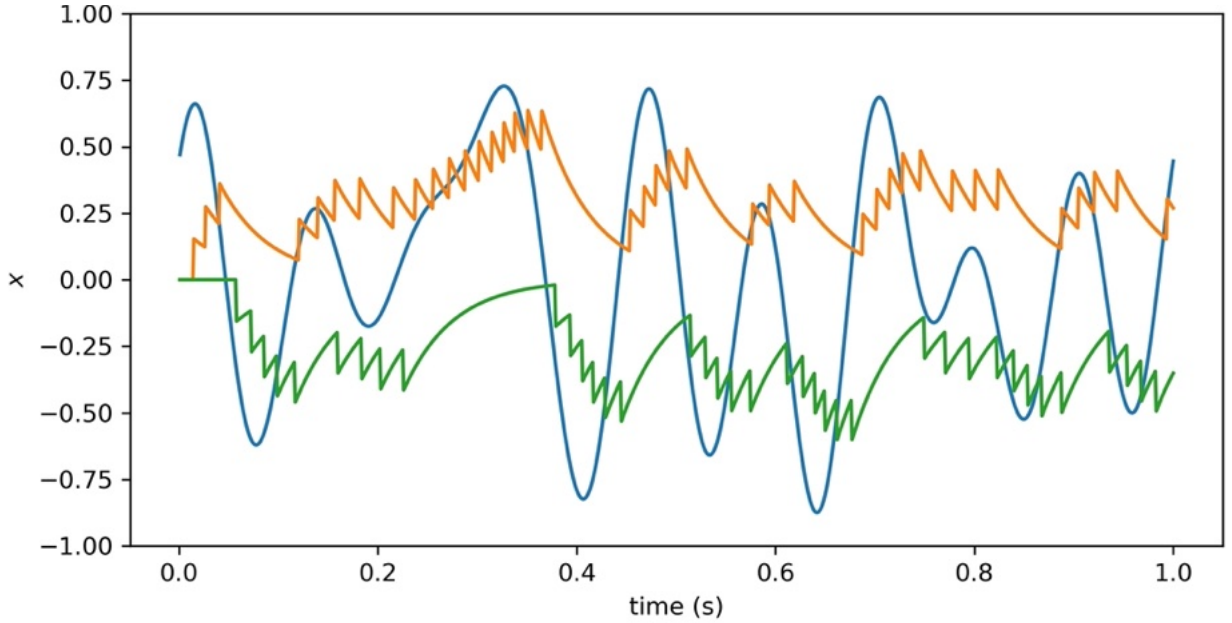
- τ is different for different neurotransmitters and different neurotransmitter receptors (10 - 150 msec)



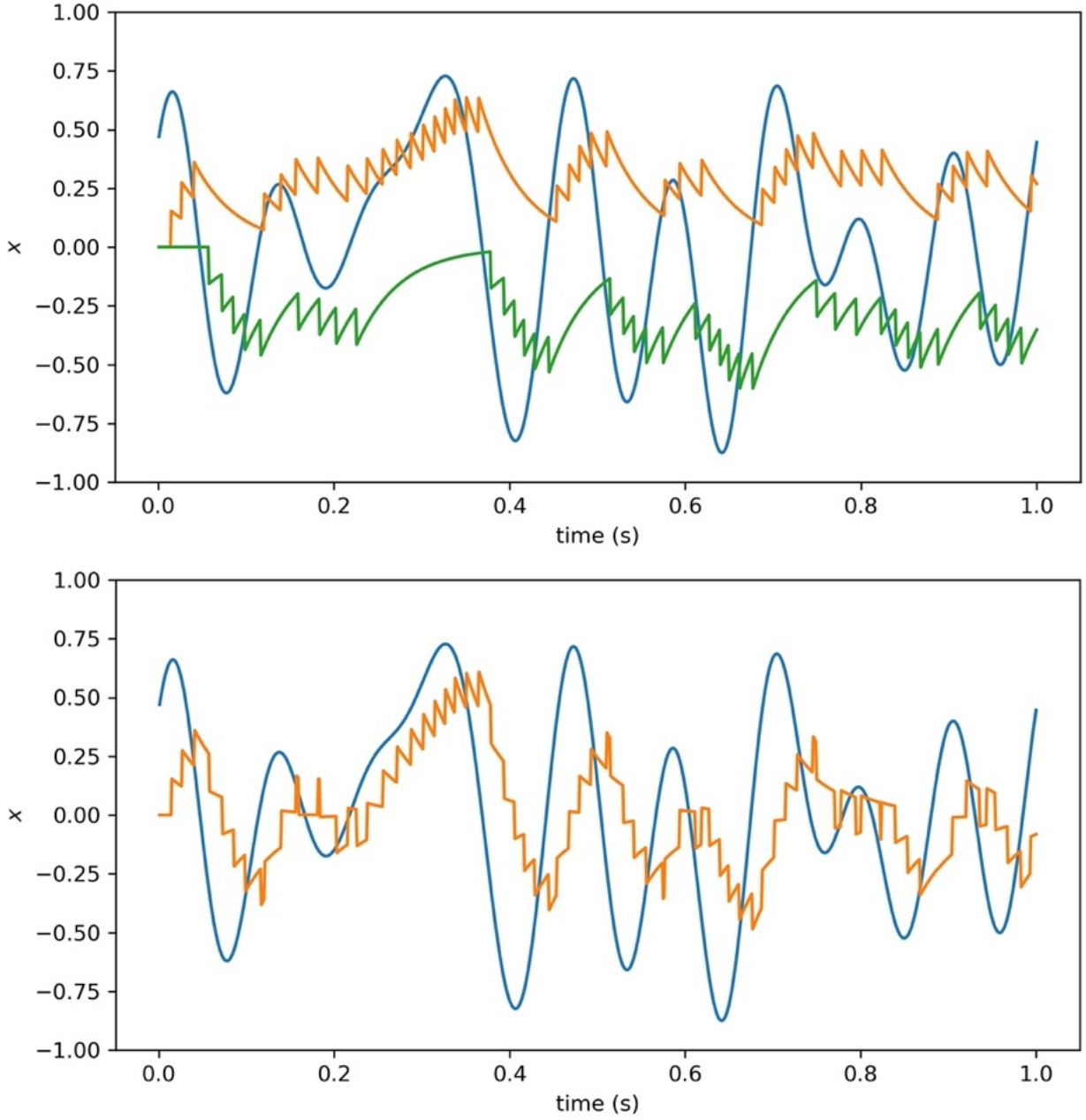
τ = 50 msec

The Neural Engineering Framework (NEF)





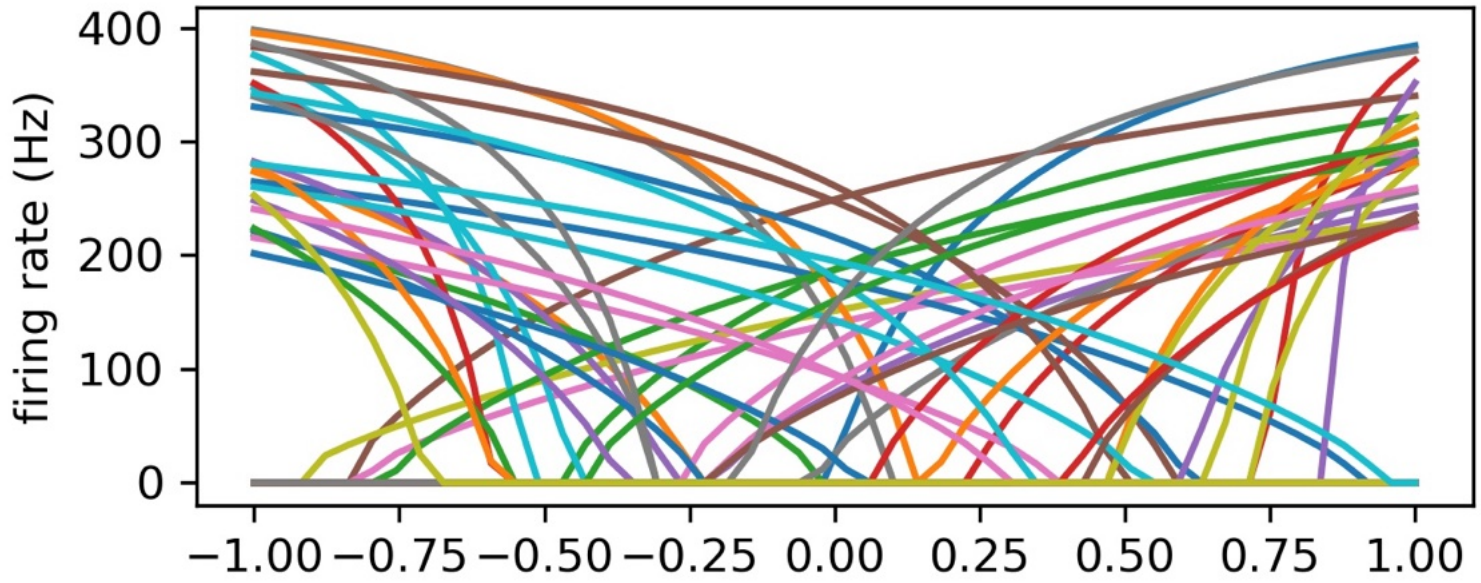
The Neural Engineering Framework (NEF)



The Neural Engineering Framework (NEF)

Representation

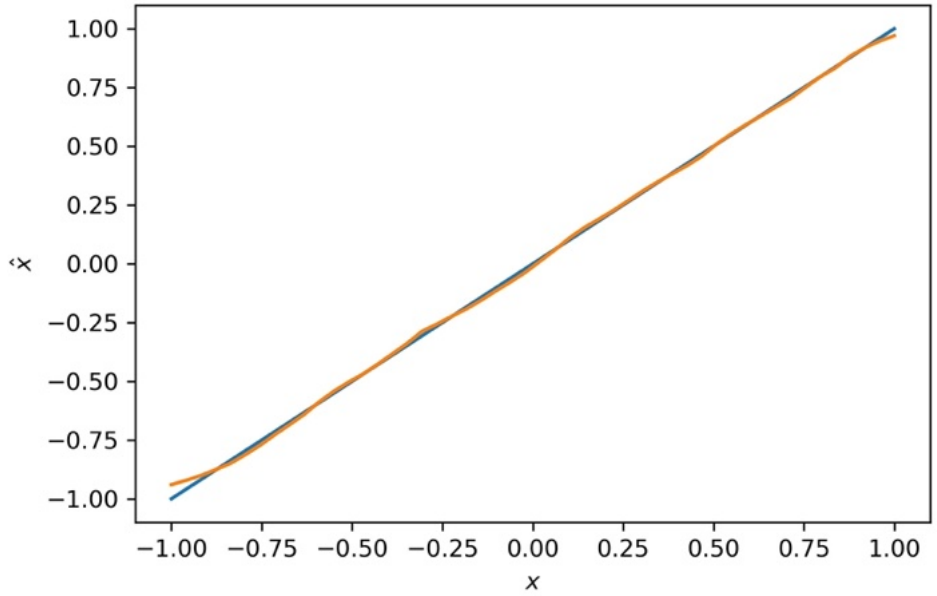
- What functions $f(x)$ can we accurately compute from a group of 50 neurons?



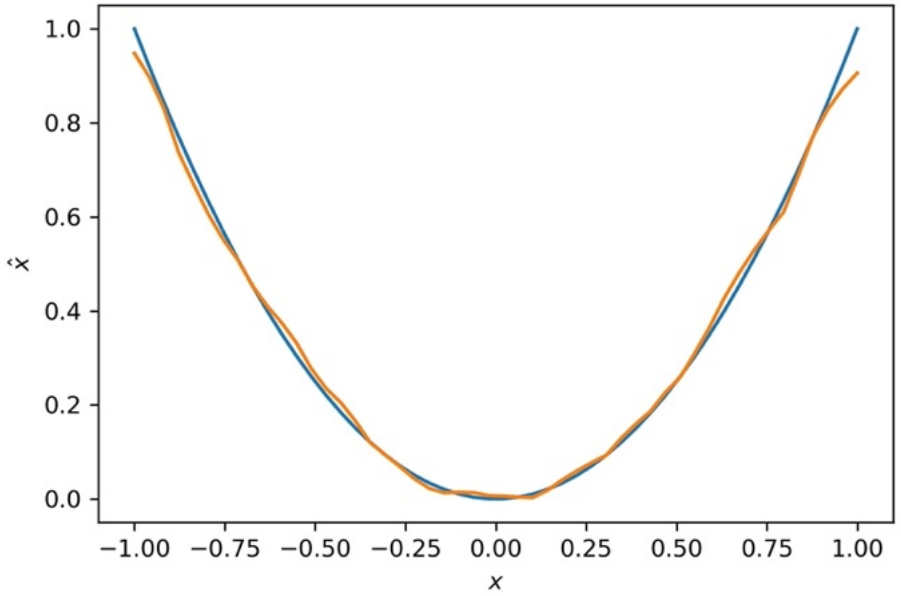
The Neural Engineering Framework (NEF)

Representation

- What functions $f(x)$ can we accurately compute from a group of 50 neurons?



f(x) = x

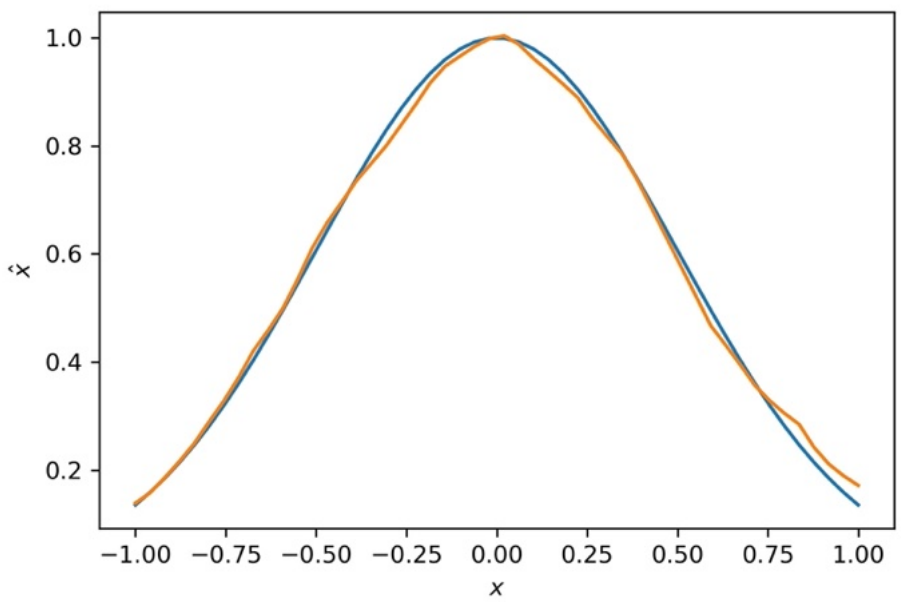


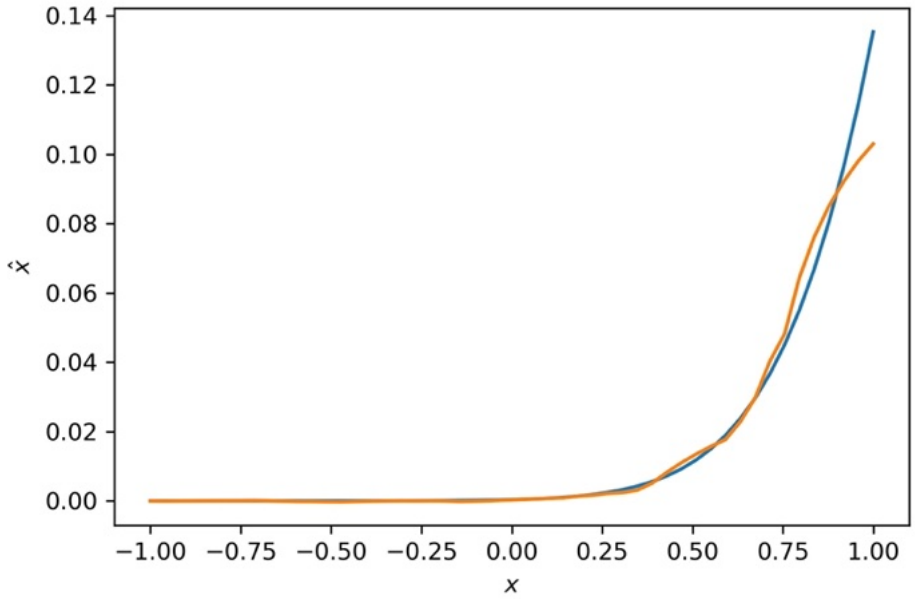
f(x) = x^2

The Neural Engineering Framework (NEF)

Representation

- What functions $f(x)$ can we accurately compute from a group of 50 neurons?





$$f(x) = e^{-(x-b)^2/(2c^2)}$$

The Neural Engineering Framework (NEF)

Representation

- Because the neural responses are non-independently driven by some variable, only a small part of the subspace of possible neural activity values is ever reached by the population.
- Γ can tells us the correlations between all neurons in the population, providing the information needed to determine that subspace

$$\sum_x a_i x/S = \sum_j \left(\sum_x a_i a_j/S \right) d_j$$

$$\Upsilon = \Gamma d$$

- With SVD (singular value decomposition) **basis functions** for the function space computable from the populations' activity A.

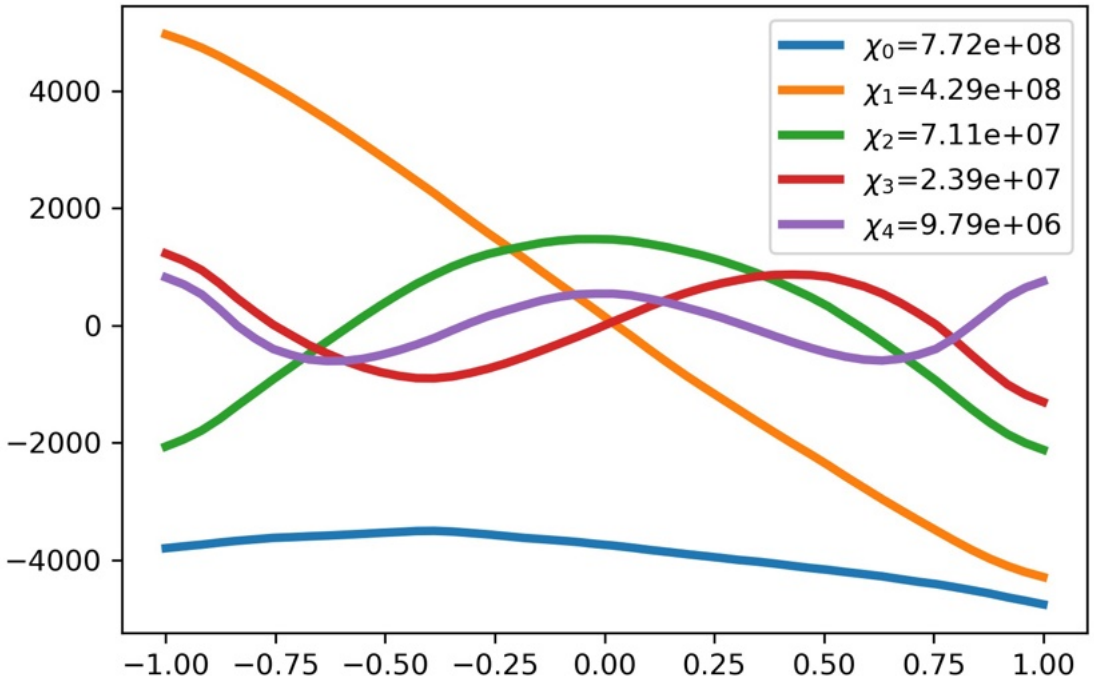
  Just as basis vectors define a vector space, so basis functions define a function space.

NOEL

The Neural Engineering Framework (NEF)

Representation

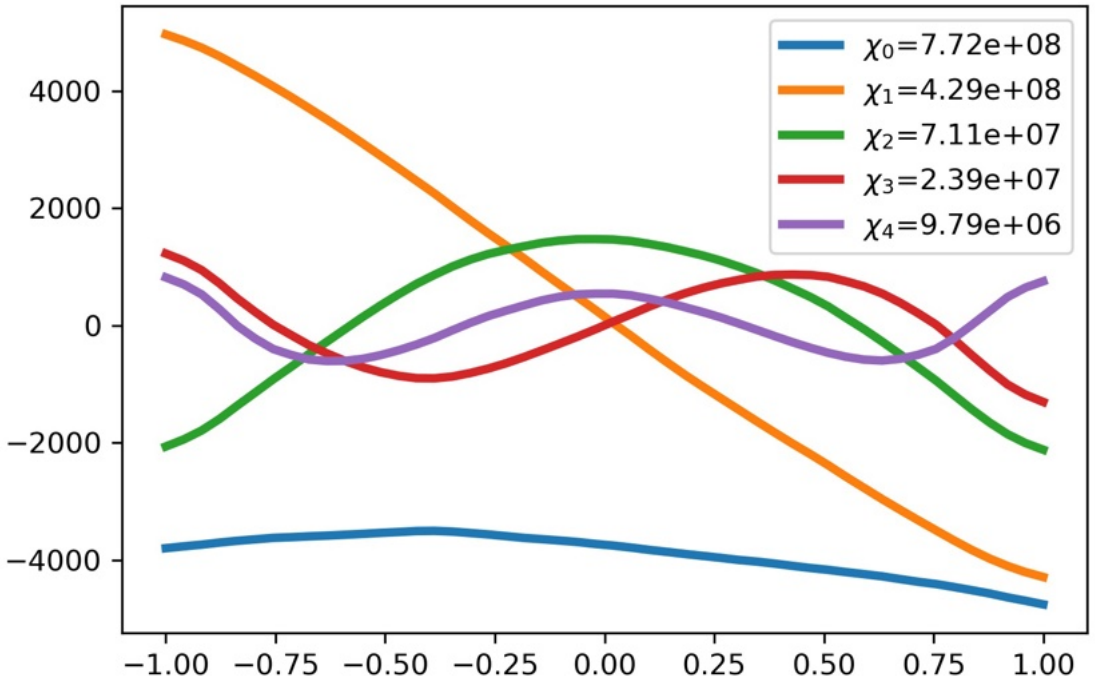
- With neural representation, the basis functions look like:

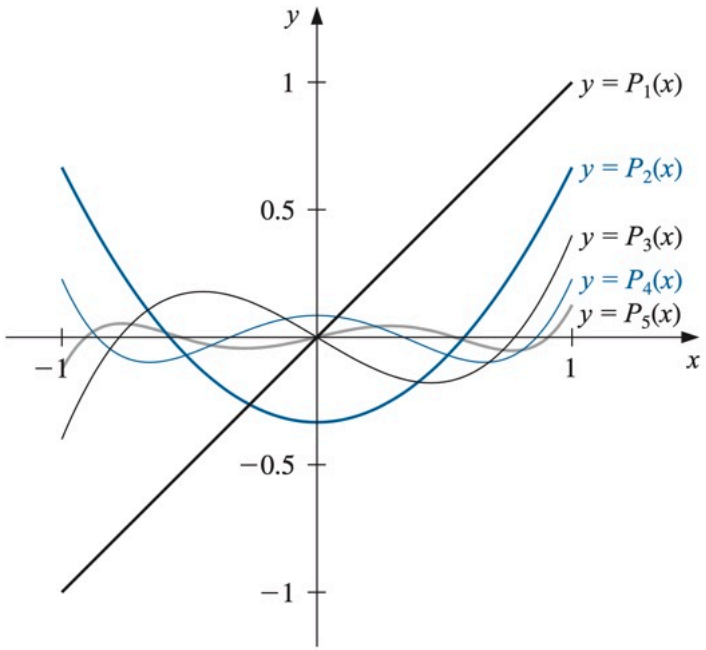


The Neural Engineering Framework (NEF)

Representation

- With neural representation, the basis functions look like:





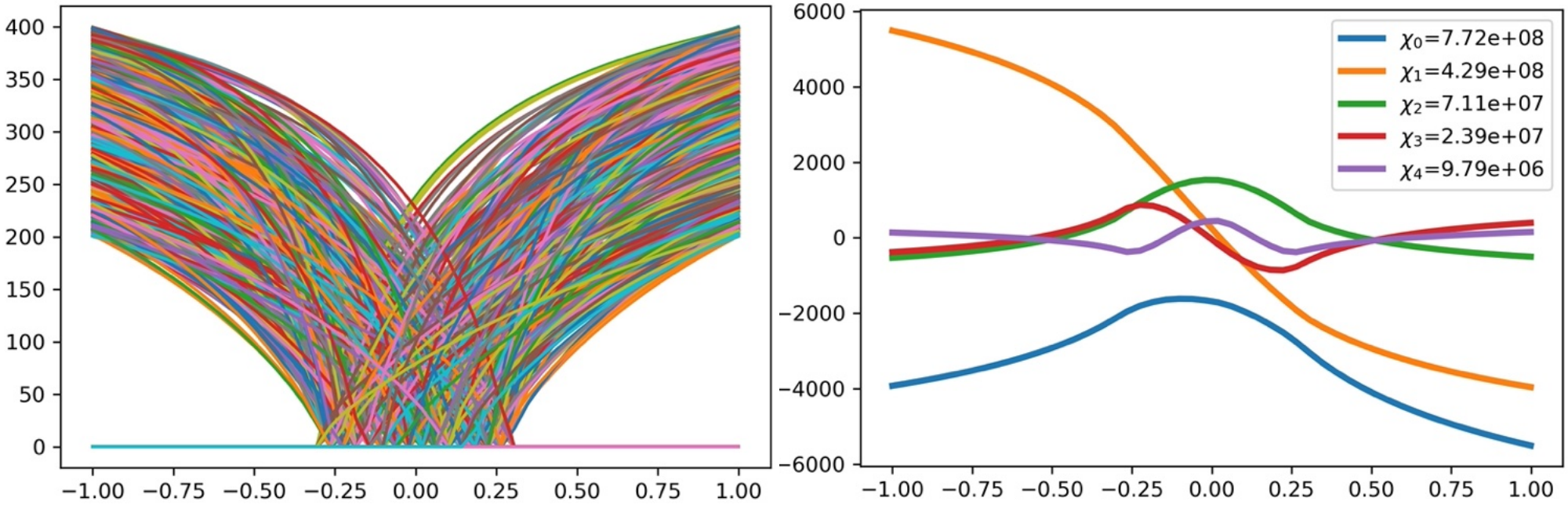
Legendre polynomials

https://en.wikipedia.org/wiki/Legendre_polynomials

The Neural Engineering Framework (NEF)

Representation

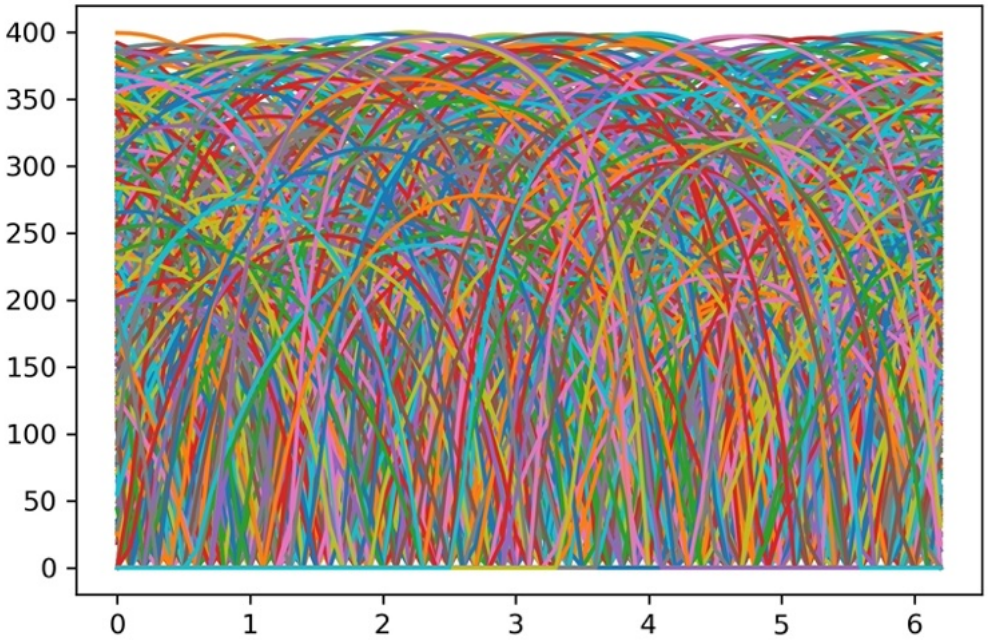
- So we should be able to approximate any function that can be approximated by polynomials
- Changing the tuning curves will change the basis functions. For example:

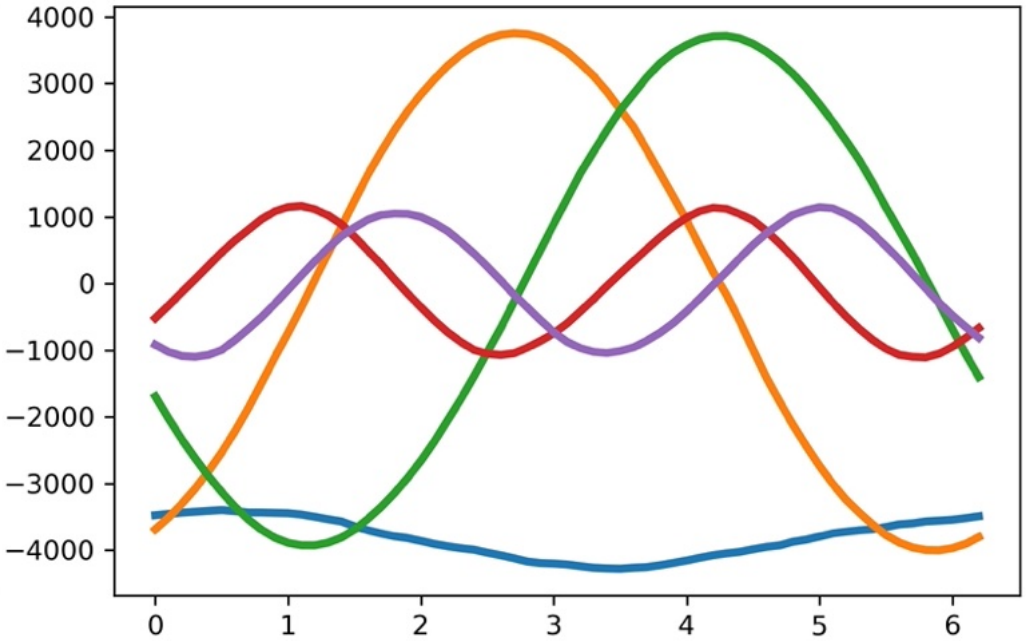


The Neural Engineering Framework (NEF)

Representation

- So we should be able to approximate any function that can be approximated by polynomials
- Changing the tuning curves will change the basis functions. For example:





NOEL

The Neural Engineering Framework (NEF)

Transformation

- So far we've just talked about neural activity in a single population
- What about connections between neurons?
- Let's say we have two groups of neurons: one group represents x, and the other group represents y
- Can we pass the value from one group of neurons to the other?

The Neural Engineering Framework (NEF)

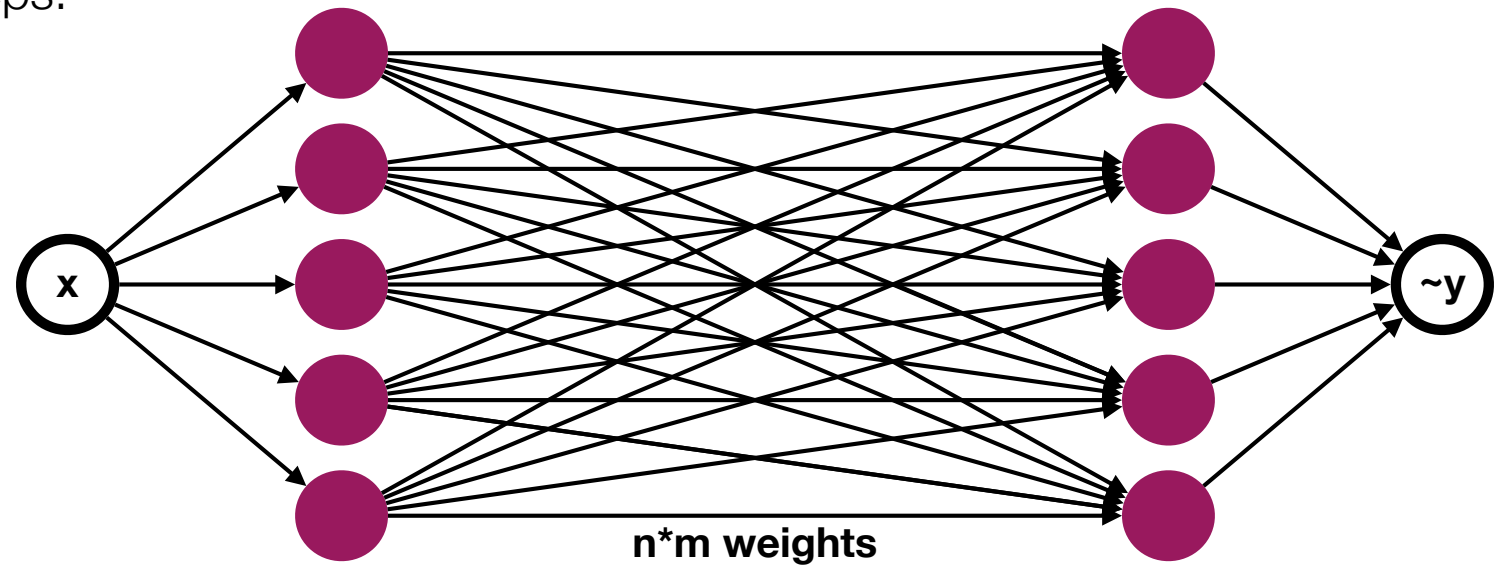
Transformation

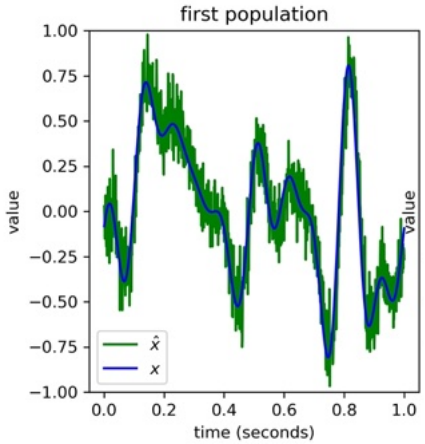
- Encoding/decoding:
  - Encode into a: $a_i = G_i[\alpha_i e_i x + J_i^{bias}]$
  - Decode from a: $\hat{x} = \sum_i a_i d_i$
  - Set $y = \hat{x}$
  - Encode into b: $b_j = G_j[\alpha_j e_j y + J_j^{bias}]$
  - Decode from b: $\hat{y} = \sum_j b_j d_j$

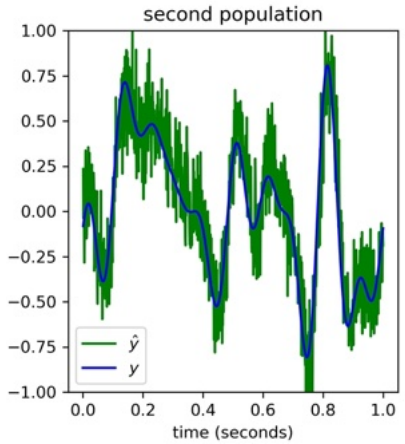
The Neural Engineering Framework (NEF)

Transformation

- Without worrying about biological plausibility, we can formulate this in two steps:







The Neural Engineering Framework (NEF)

Transformation

- Encoding/decoding:
  - Encode into a: $a_i = G_i[\alpha_i e_i x + J_i^{bias}]$
  - Decode from a: $\hat{x} = \sum_i a_i d_i$
  - Set $y = \hat{x}$
  - Encode into b: $b_j = G_j[\alpha_j e_j y + J_j^{bias}]$
  - Decode from b: $\hat{y} = \sum_j b_j d_j$

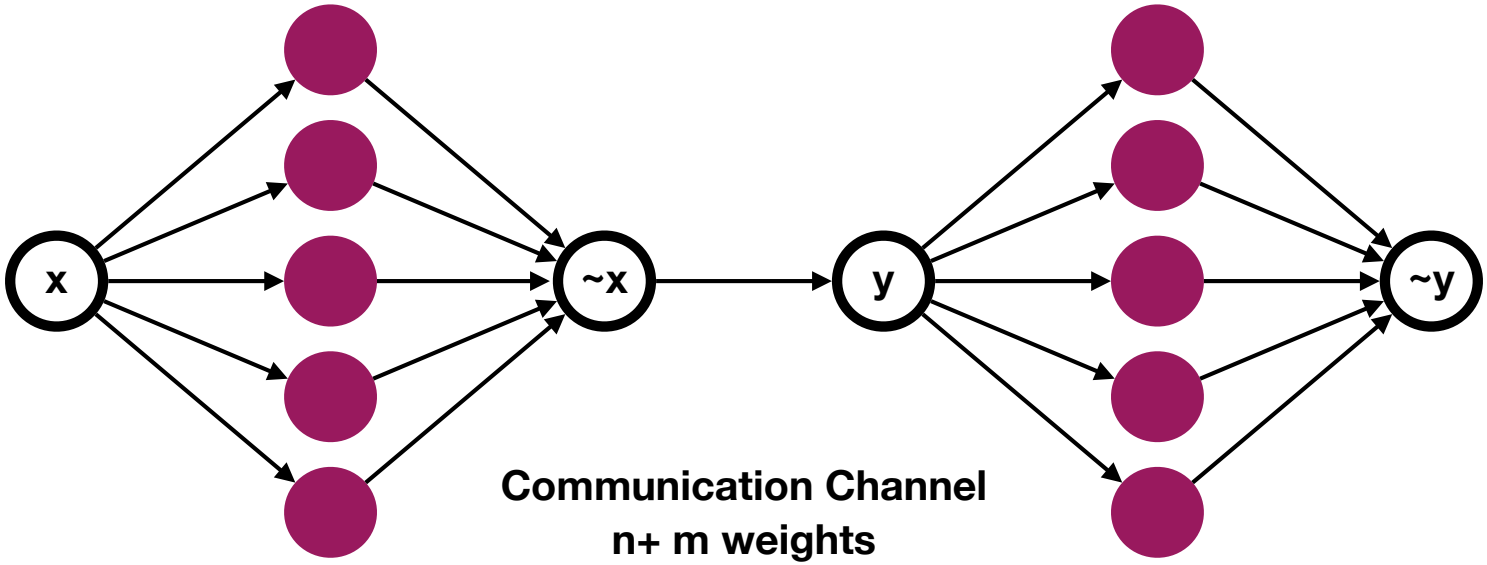
$$\mathbf{u} \otimes \mathbf{v} = \mathbf{A} = \begin{bmatrix} u_1 v_1 & u_1 v_2 & \dots & u_1 v_n \\ u_2 v_1 & u_2 v_2 & \dots & u_2 v_n \\ \vdots & \vdots & \ddots & \vdots \\ u_m v_1 & u_m v_2 & \dots & u_m v_n \end{bmatrix}$$

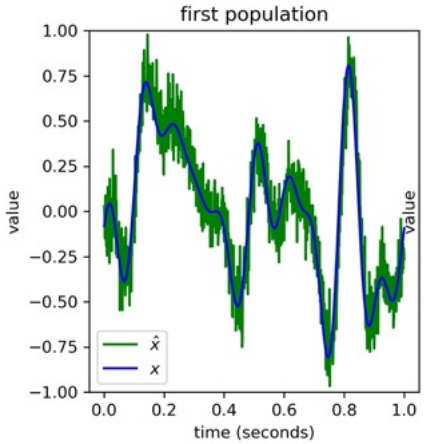
- Substitution:
  - I.e. substitute $y = \hat{x} = \sum_i a_i d_i$ into b
  - $b_j = G_j[\alpha_j e_j \sum_i a_i d_i + J_j^{bias}]$
  - $b_j = G_j[\sum_i \alpha_j e_j d_i a_i + J_j^{bias}]$
  - $b_j = G_j[\sum_i \omega_{ij} a_i + J_j^{bias}]$
  - where $\omega_{ij} = \alpha_j e_j \cdot d_i$ (an outer product)

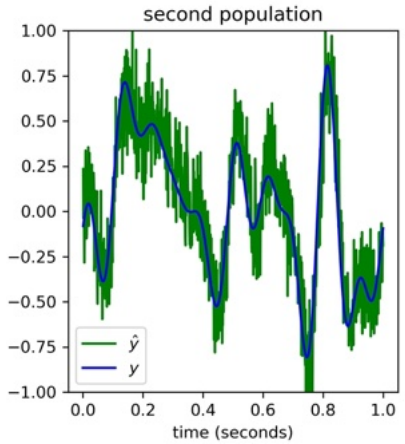
- We can get the entire weight matrix just by multiplying the decoders from the first population with the encoders from the second population

The Neural Engineering Framework (NEF)

Transformation







Neuro & Biomorphic Engineering LAB

The Neural Engineering Framework (NEF)

Transformation

- Instead of computing ωij at all, it is (usually) more efficient to just do the encoding/decoding
  - These are numerically identical processes
  - Saves memory space, no need to store a giant weight matrix
  - NxM multiplies for weights, and only ~N+M multiplies for encode/decode
- What about transforming that information in some way?
- Instead of y = x, can we get: y = f(x), e.g. $y= x^2$?

The Neural Engineering Framework (NEF)

Transformation

- Let's try $y=2x$ to start
- We already have a decoder for x, so how do we get a decoder for 2x?
- Two ways:
  - Multiply the 'representational' decoder by 2
  - Use 2x when computing Υ

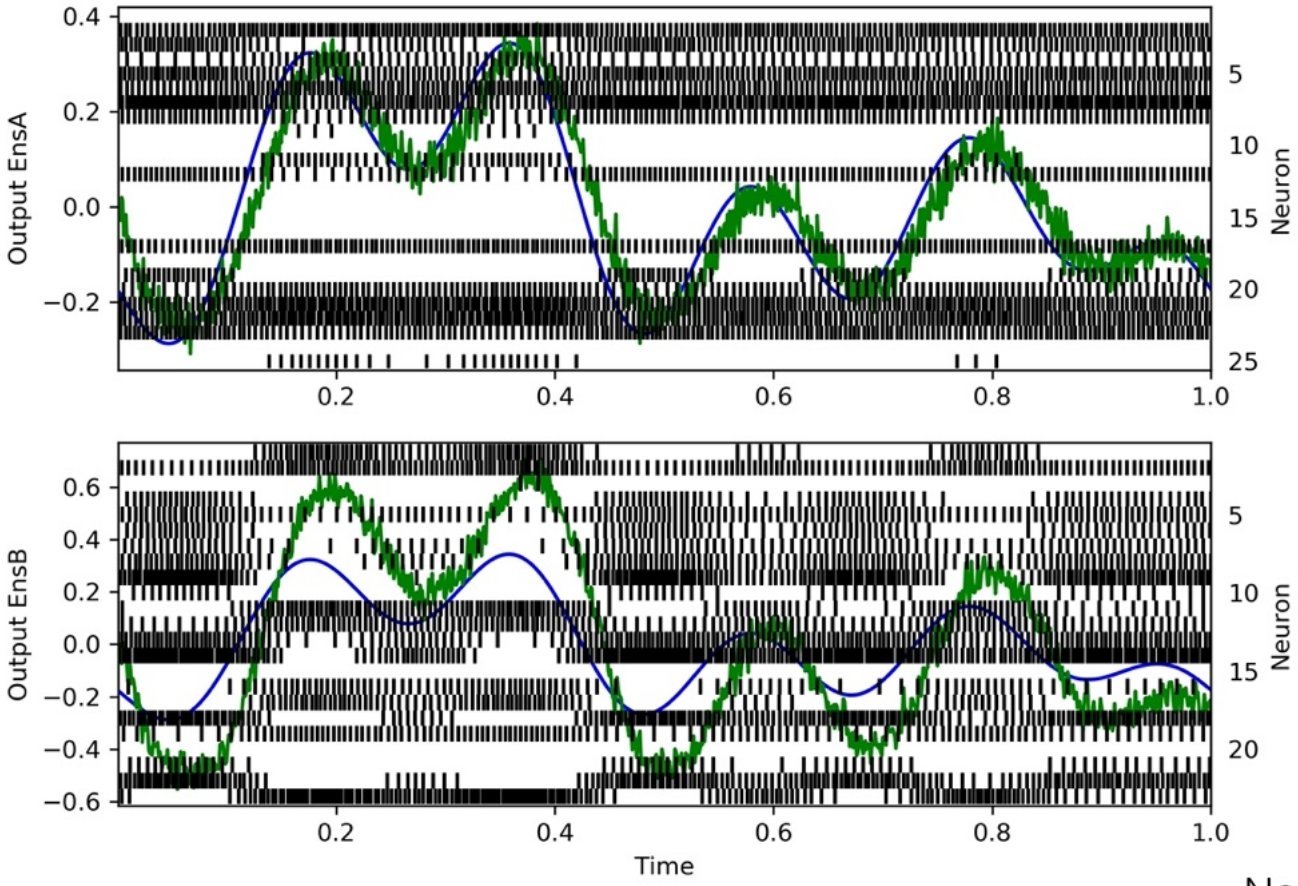
- In equations: $\Upsilon = \Gamma d \rightarrow d = \Gamma^{-1}\Upsilon \rightarrow d_i = \sum_j \Gamma^{-1}_{ij}\Upsilon_j$
  - $d^{f(x)} = \Gamma^{-1}\Upsilon^{f(x)}$
  - $\Upsilon_i^{f(x)} = \sum_x a_i f(x)\, dx$
  - $\Gamma_{ij} = \sum_x a_i a_j\, dx$
  - $\hat{f}(x) = \sum_i a_i d_i^{f(x)}$

  We call standard d_i “representational decoders”
- We call $d_i^{f(x)}$ "transformational decoders" (or "decoders for f(x)“)

The Neural Engineering Framework (NEF)

Transformation

- Let's try $y=2x$ to start. If you already have a decoder for x, you can quickly find a decoder for any linear function of x
- If the decoder for x is d, the decoder for Mx is Md

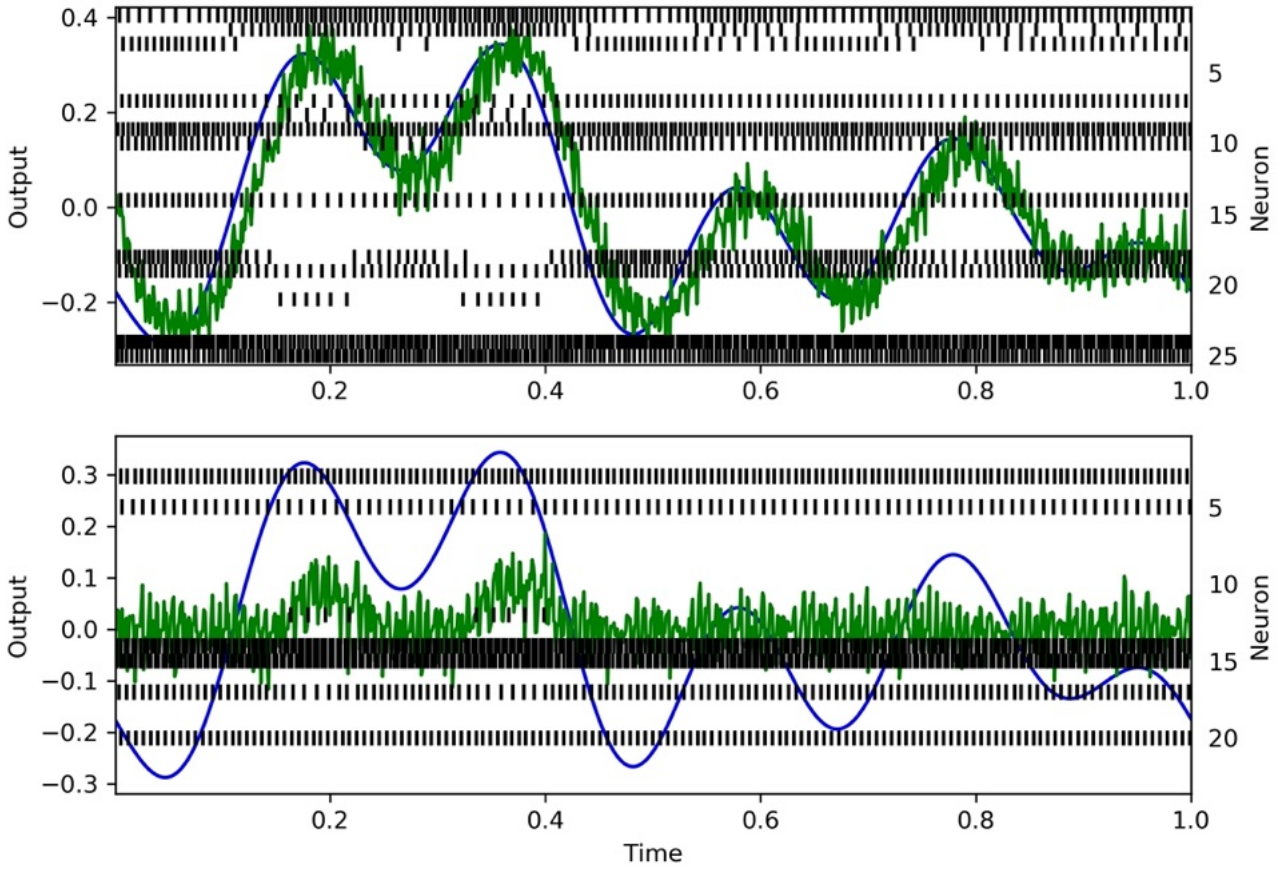


Neuro & Biomorphic Engineering LAB

The Neural Engineering Framework (NEF)

Transformation

- What about $y=x^2$?
- For some other function of x, substitute in function f(x) when finding Y



Neuro & Biomorphic Engineering LAB

The Neural Engineering Framework (NEF)

Transformation

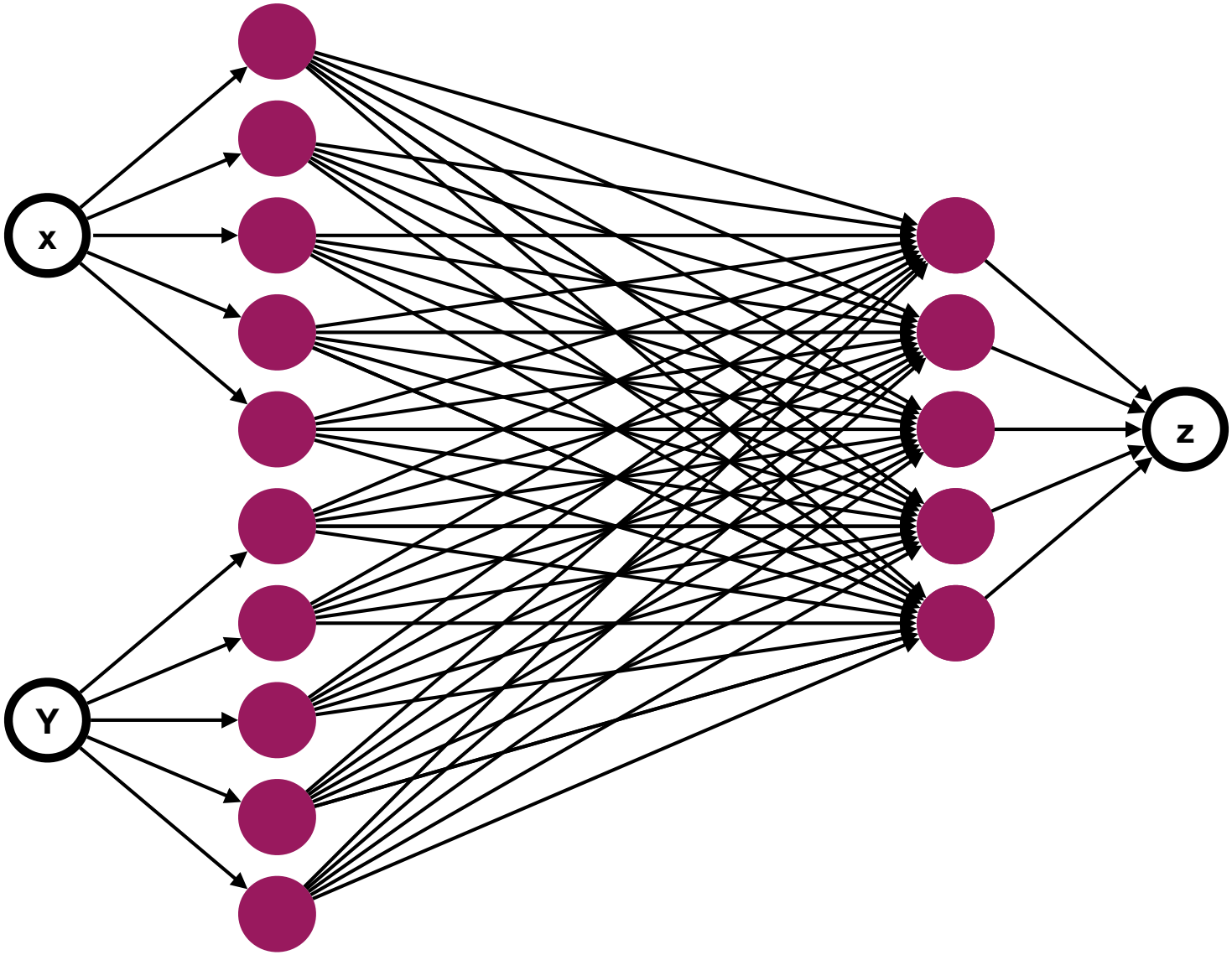
- **Addition**: We want the total current going into a z neuron to be:

$$J = \alpha e \cdot (x + y) + J^{bias}$$

- Substitute into the equation: $z = x + y \approx \hat{x} + \hat{y}$
  - $J_k = \alpha_k e \cdot (\hat{x} + \hat{y}) + J_k^{bias}$
  - $\hat{x} = \sum_i a_i d_i$
  - $\hat{y} = \sum_j a_j d_j$
  - $J_k = \alpha_k e_k \cdot (\sum_i a_i d_i + \sum_j a_j d_j) + J_k^{bias}$
  - $J_k = \sum_i (\alpha_k e_k \cdot d_i a_i) + \sum_j (\alpha_k e_k \cdot d_j a_j) + J_k^{bias}$
  - $J_k = \sum_i (\omega_{ik} a_i) + \sum_j (\omega_{jk} a_j) + J_k^{bias}$
  - $\omega_{ik} = \alpha_k e_k \cdot d_i$ and $\omega_{jk} = \alpha_k e_k \cdot d_j$

The Neural Engineering Framework (NEF)

Transformation

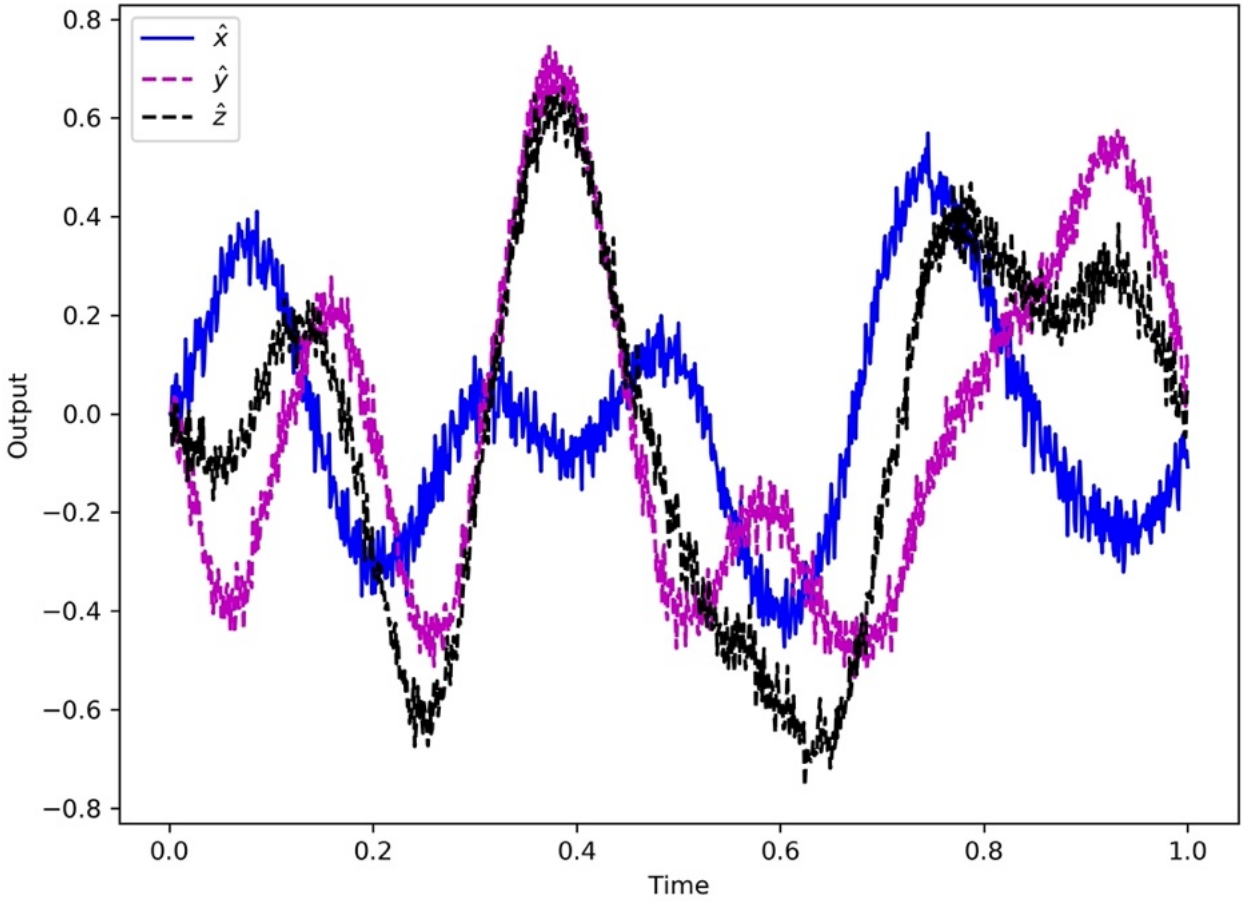


The Neural Engineering Framework (NEF)

Transformation

- Putting multiple inputs into a neuron automatically gives us addition!
- Addition: We want the total current going into a z neuron to be:

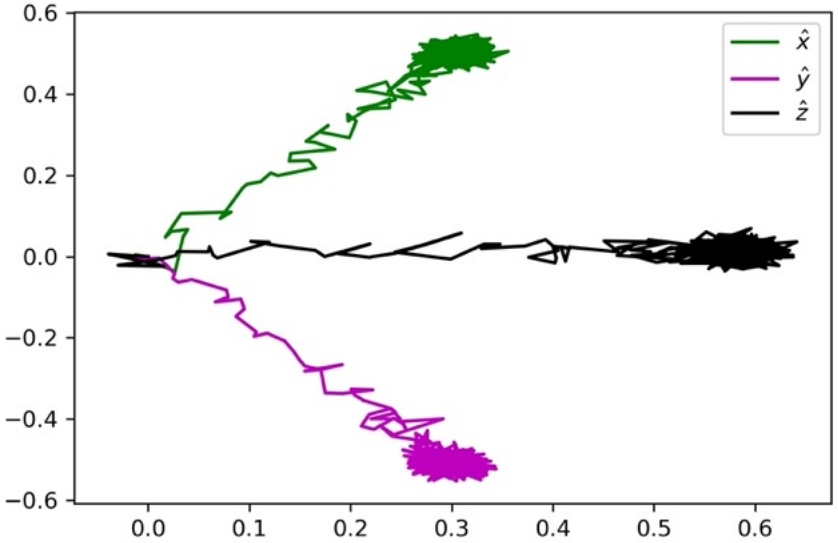
$$J = \alpha e \cdot (x + y) + J^{bias}$$

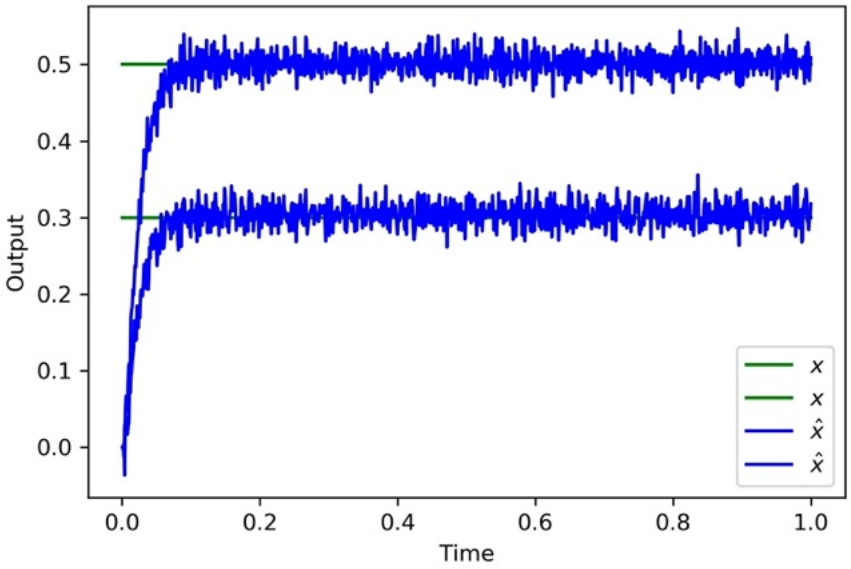


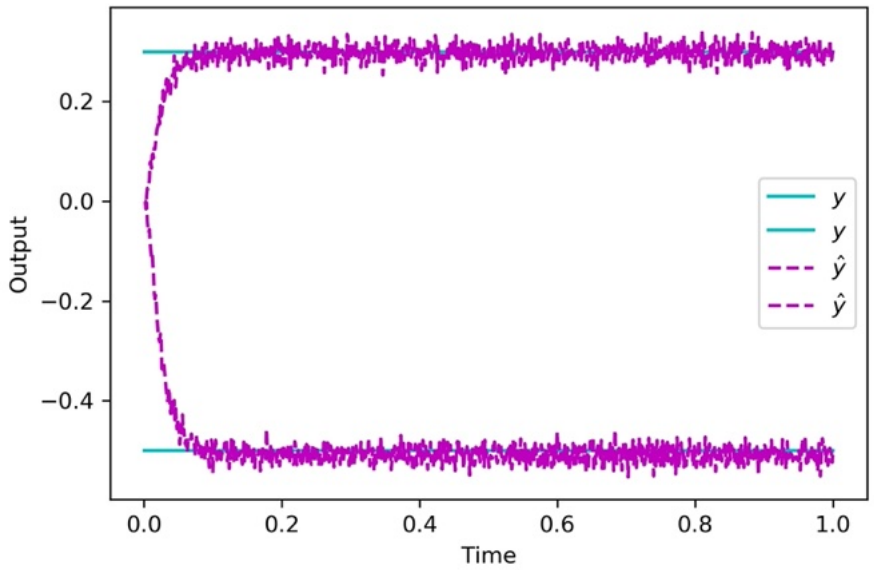
The Neural Engineering Framework (NEF)

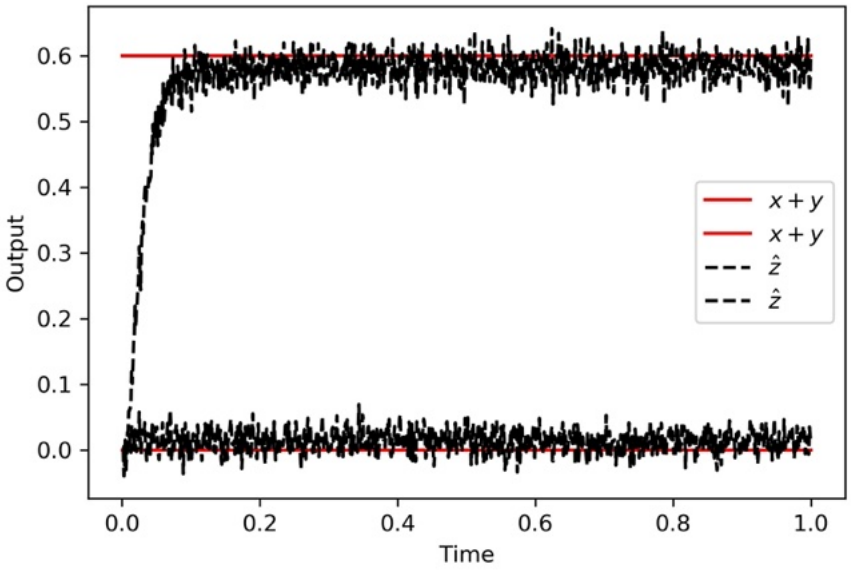
Transformation

- Addition: adding two vectors









The Neural Engineering Framework (NEF)

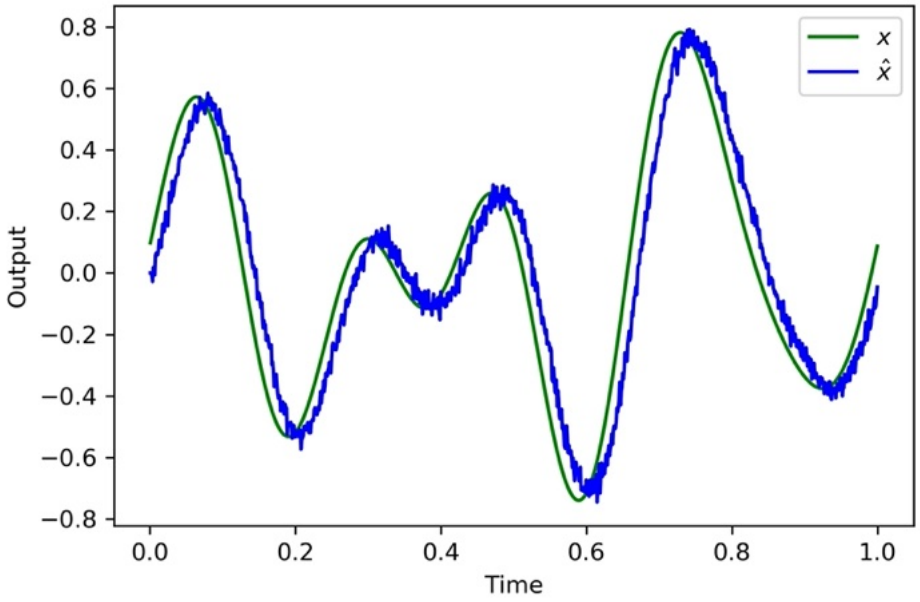
Transformation

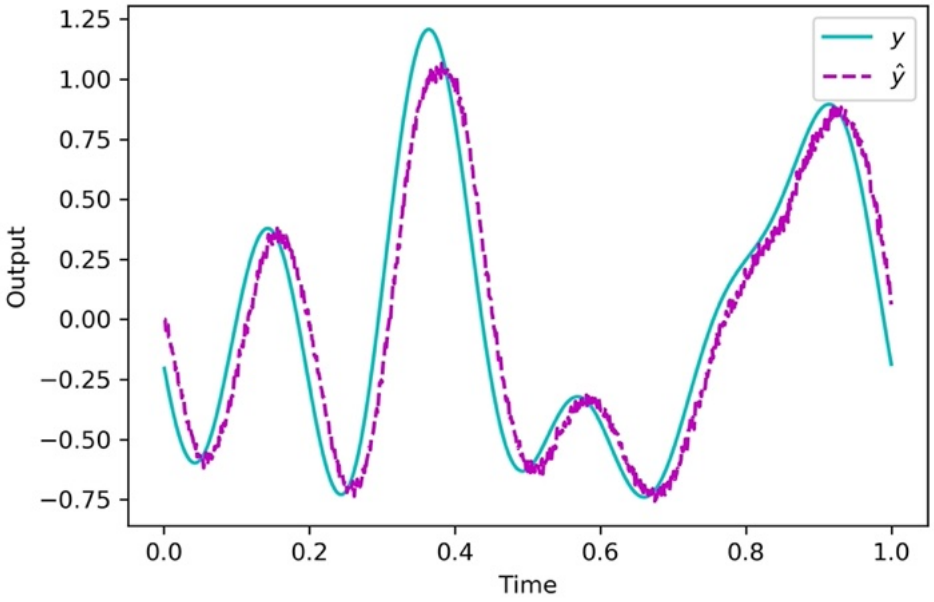
- What if we want to combine to compute a nonlinear function of two inputs?
- E.h. $z = x \times y$
- We know how to compute nonlinear functions of a vector space (E.g. x^2)
- If x is a vector, you get a bunch of cross terms
- E.g. if x is 2D this gives $x_1^2 + 2x_1x_2+x_2^2$
- This means that if you combine two inputs into a 2D space, you can get out their product

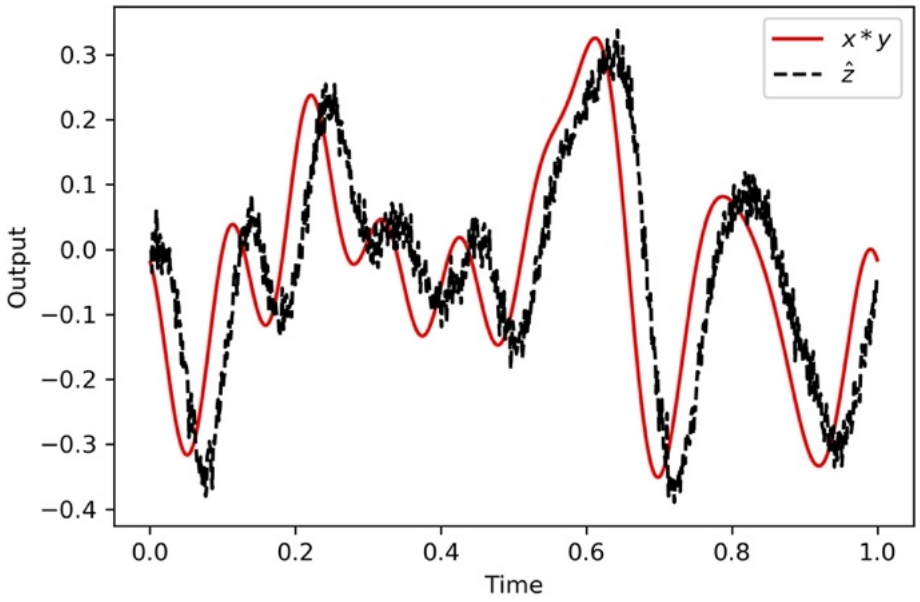
The Neural Engineering Framework (NEF)

Transformation

- Multiplying two inputs







Multiplication is quite powerful, and has lots of uses:

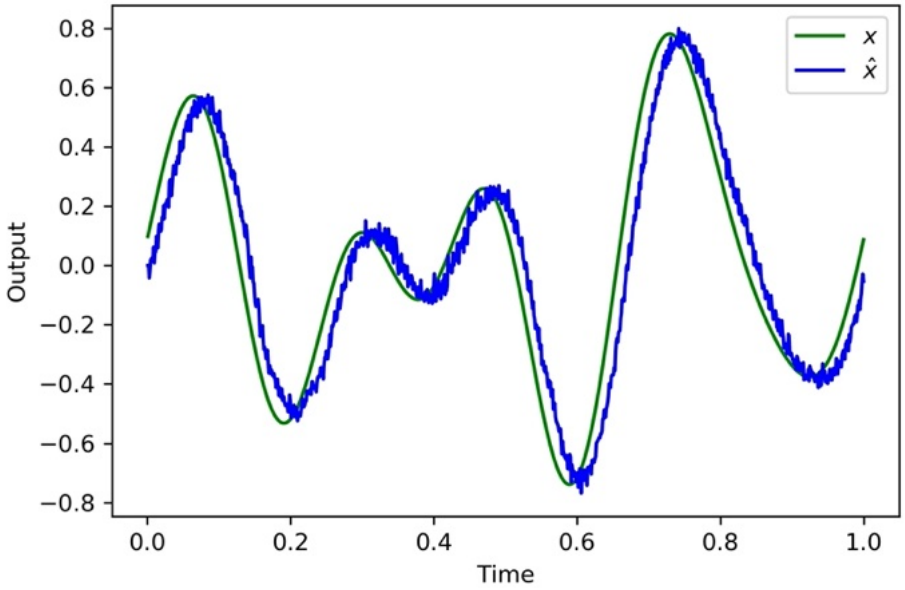
- Gating of signals
- Attention effects
- Binding
- Statistical inference

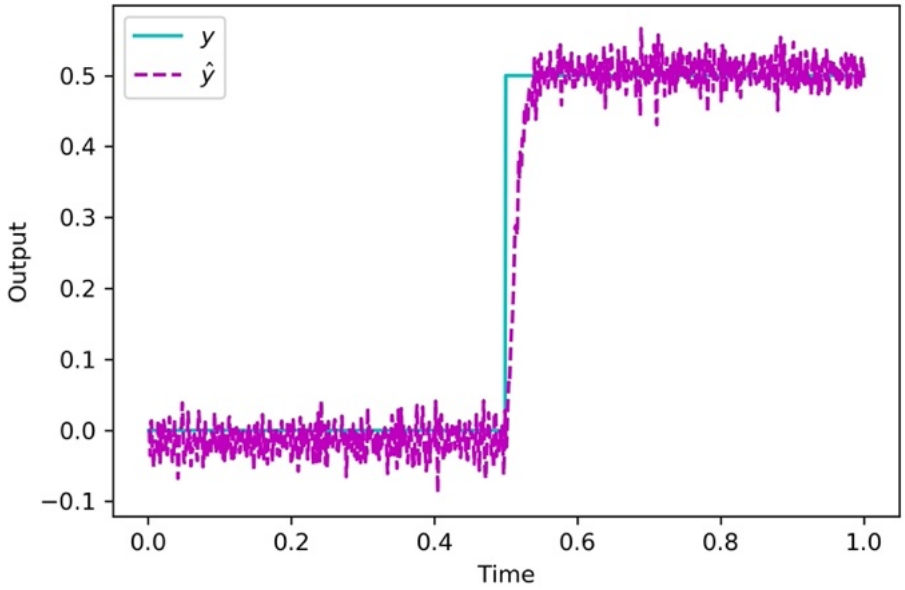
Neuro & Biomorphic Engineering LAB

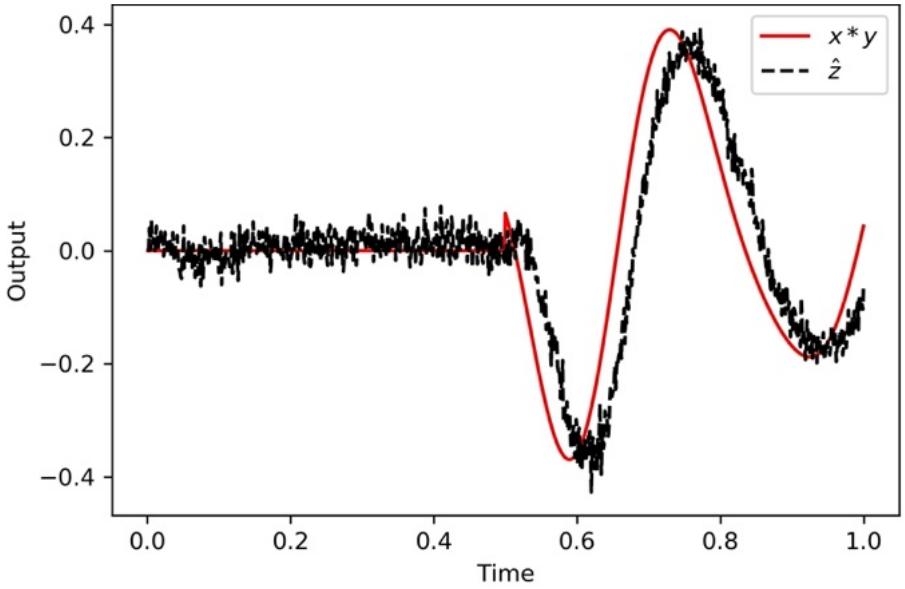
The Neural Engineering Framework (NEF)

Transformation

- Gating example





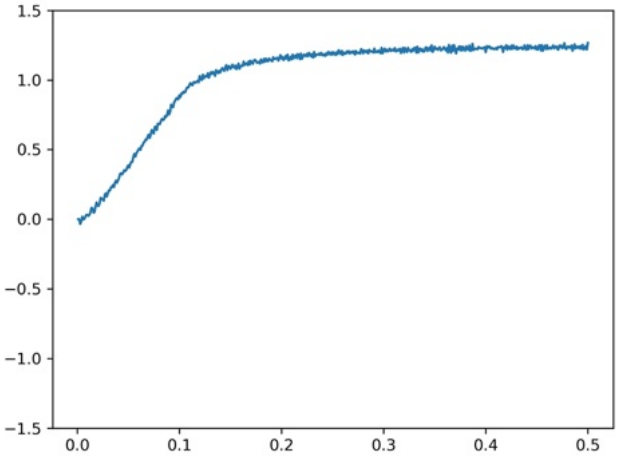


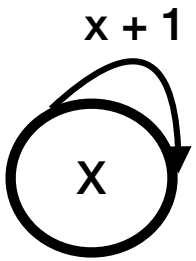
Neuro & Biomorphic Engineering LAB

The Neural Engineering Framework (NEF)

Dynamics

- Everything we've looked at so far has been feedforward
  - There's some pattern of activity in one group of neurons representing x
  - We want that to cause some pattern of activity in another group of neurons to represent $y=f(x)$
  - These can be chained together to make more complex systems $z=h(f(x)+g(y))$
- What about recurrent networks?
  - What happens when we connect a neural group back to itself?
  - Consider: $x_{t+1} = x_t + 1$



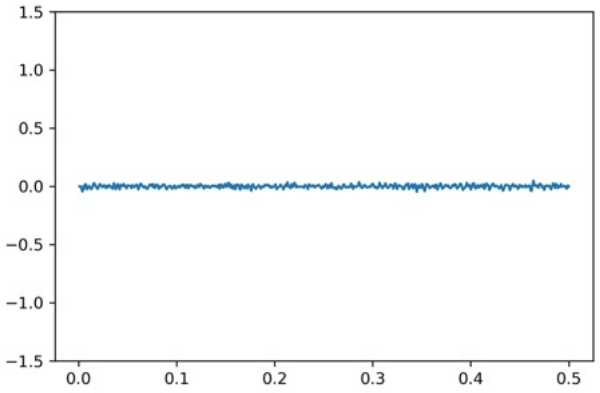


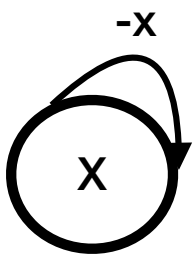

```
ensA = nengo.Ensemble(100, dimensions=1)
    def feedback(x):
        return x+1

    conn = nengo.Connection(ensA, ensA,
                function=feedback, synapse = 0.1)

    ensA_p = nengo.Probe(ensA, synapse=.01)
```

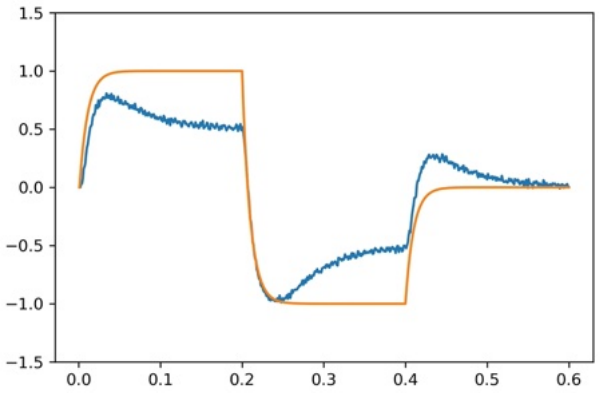
The Neural Engineering Framework (NEF)

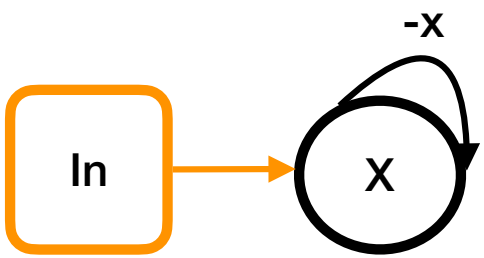
Dynamics

- f(x) = -x





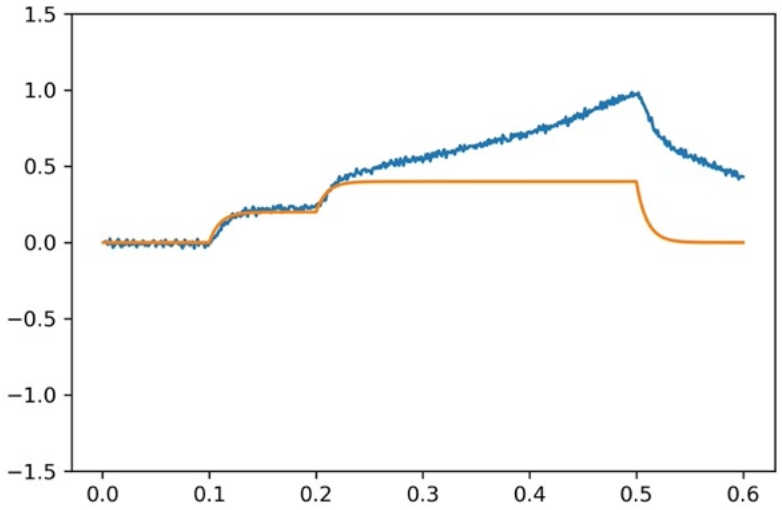


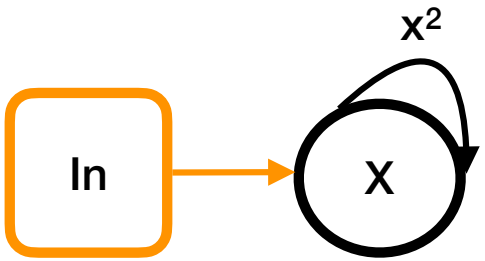


The Neural Engineering Framework (NEF)

Dynamics

- $f(x) = x^2$

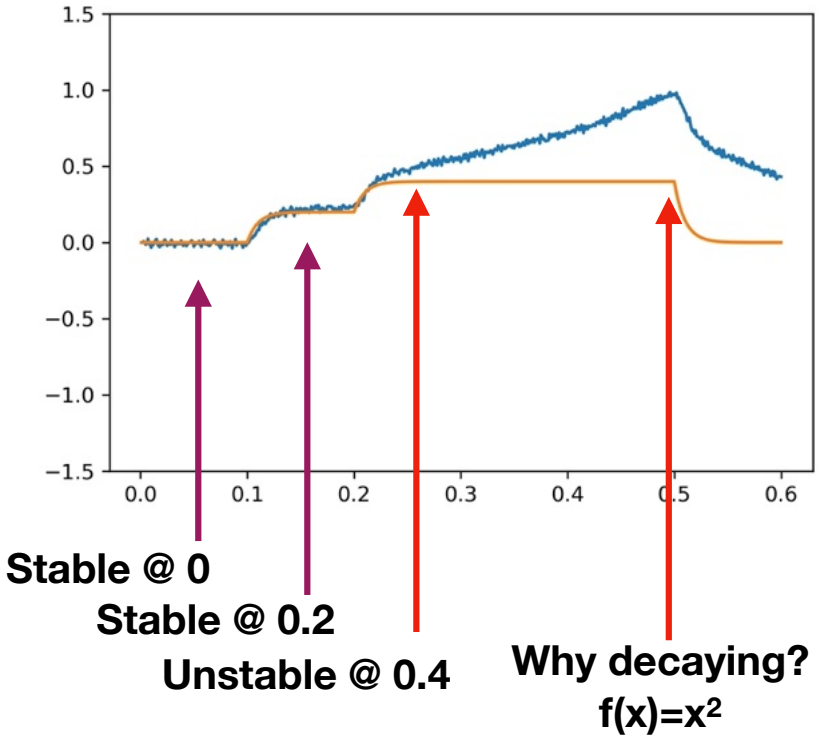


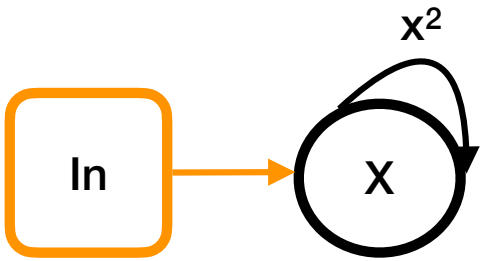


The Neural Engineering Framework (NEF)

Dynamics

- $f(x) = x^2$

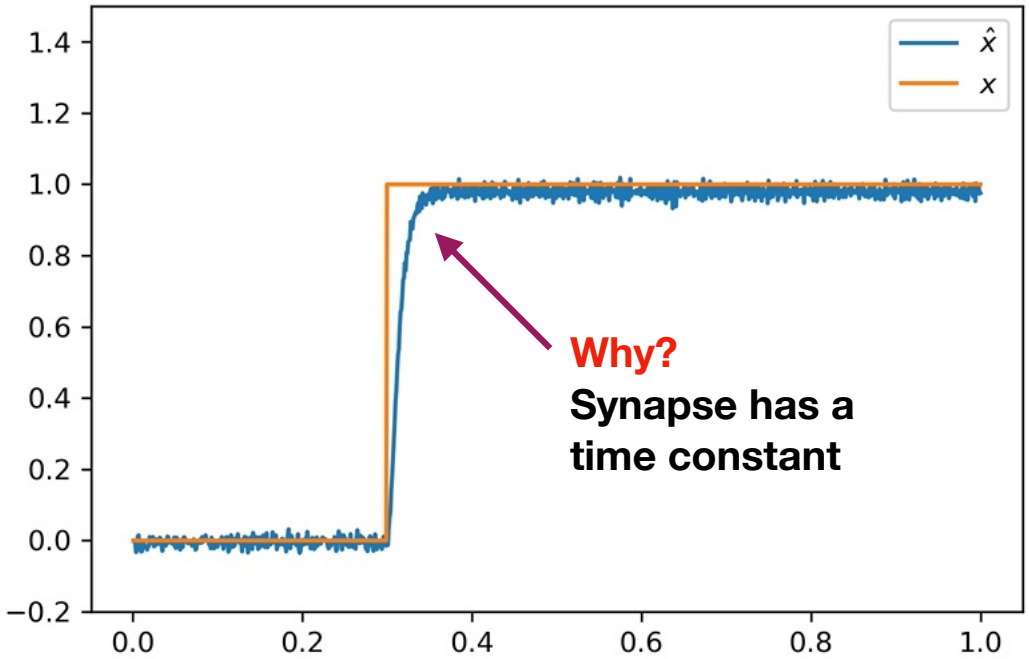




The Neural Engineering Framework (NEF)

Dynamics

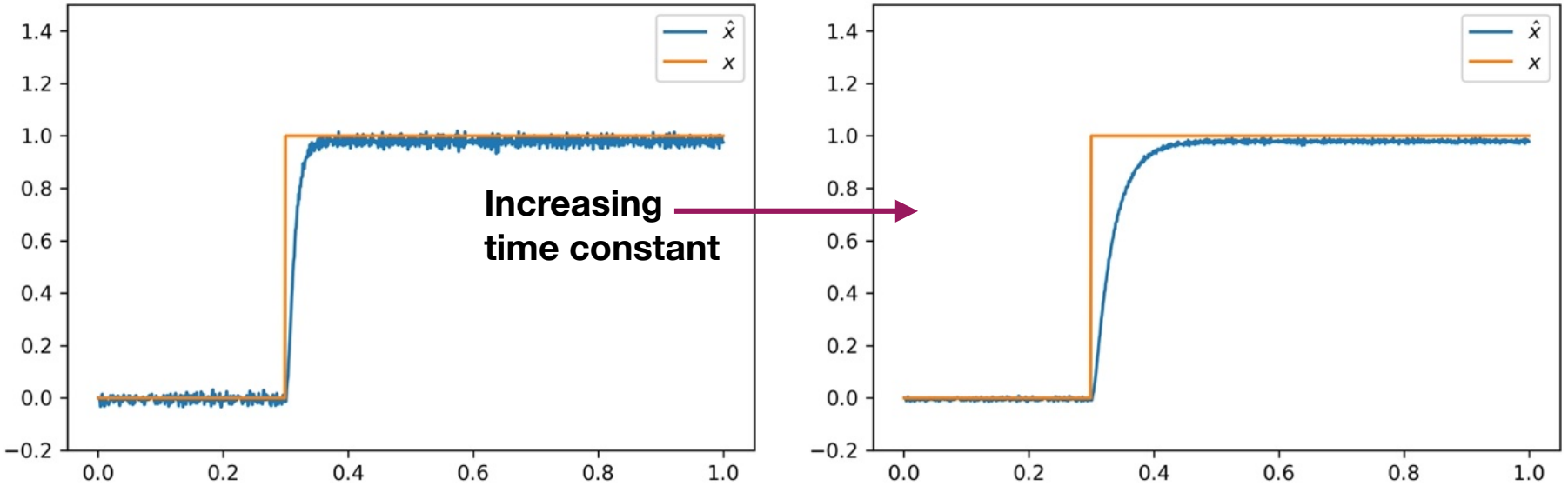
- Going back to the basics
- Just a single feed-forward neural population: encode x into current, compute spikes, decode filtered spikes into ~x
- Instead of a constant input, let's change the input from zero to one



The Neural Engineering Framework (NEF)

Dynamics

- Going back to the basics
- Just a single feed-forward neural population: encode x into current, compute spikes, decode filtered spikes into ~x
- Instead of a constant input, let's change the input from zero to one



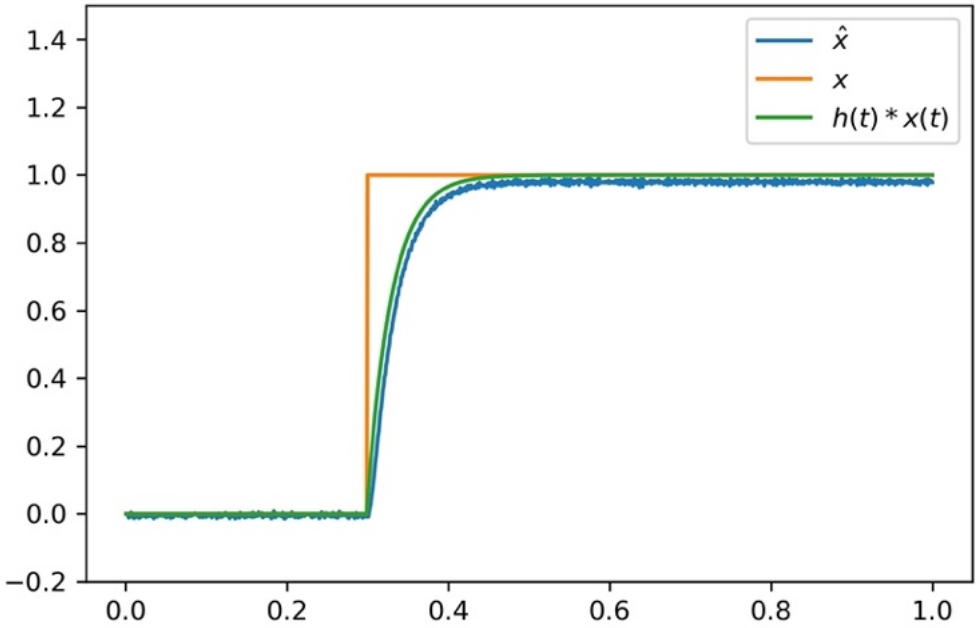
The Neural Engineering Framework (NEF)

Dynamics

- Instead of getting f(x) = x, we have f(x(t)) = x(t) * h(t), where h(t) is a low pass filter:

$$h(t) = \begin{cases} e^{-t/\tau} & \text{if } t > 0 \\ 0 & \text{otherwise} \end{cases}$$

- Recurrent connections are dynamic as well (i.e. passing past information to future state of the population)



The Neural Engineering Framework (NEF)

Dynamics

Recurrent connections

- A connection actually approximates f(x(t)) * h(t)
- A recurrent connection: x(t) = f(x(t)) * h(t)
- Convolutions are hard to figure out.
- We could do a Fourier transform (as we previously did): $X(\omega) = F(\omega)H(\omega)$
- Since we are studying the response of a system (rather than a continuous signal), there's a more general and appropriate transform
- Laplace transform (https://en.wikipedia.org/wiki/Laplace_transform)
- For our equations: $X(s) = F(s)H(s)$

  $$H(s) = \frac{1}{1+s\tau}$$

  where *s* is a complex number frequency parameter: $s = \sigma + i\omega$

The Neural Engineering Framework (NEF)

Dynamics

Recurrent connections

- Laplace transform: $X(s) = F(s)H(s)$

$$H(s) = \frac{1}{1+s\tau}$$

- Rearranging:

$$X(s) = F(s)\frac{1}{1+s\tau}$$

$$X(s)(1 + s\tau) = F(s)$$

$$X(s) + X(s)s\tau = F(s)$$

$$sX(s) = \frac{1}{\tau}(F(s) - X(s))$$

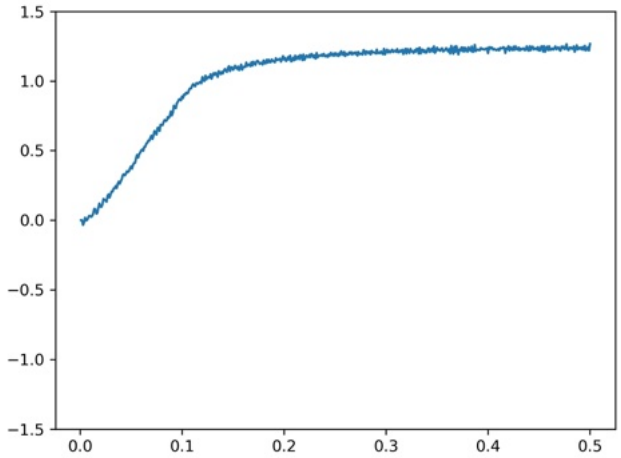
- Convert back into the time domain (inverse Laplace):

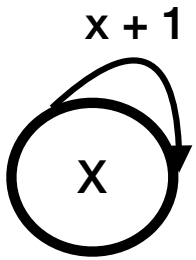
$$\frac{dx}{dt} = \frac{1}{\tau}(f(x(t)) - x(t))$$

- And so if we introduce a recurrent connection, we are implementing a differential equation

The Neural Engineering Framework (NEF)

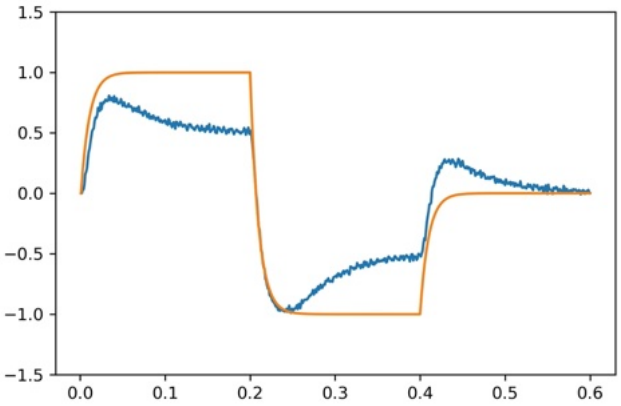
Dynamics

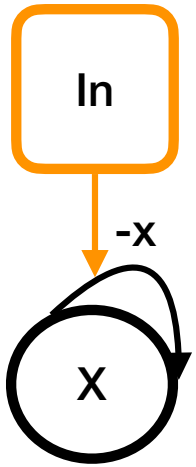




$$f(x) = x + 1$$

- $\dot{x} = \frac{1}{\tau}(x + 1 - x)$
- $\dot{x} = \frac{1}{\tau}$





$$f(x) = -x?$$

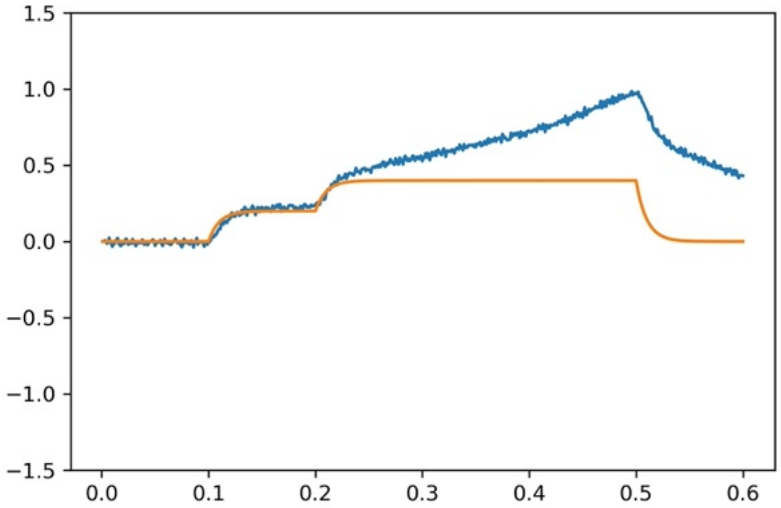
- $\dot{x} = \frac{1}{\tau}(-x - x)$
- $\dot{x} = \frac{-2x}{\tau}$

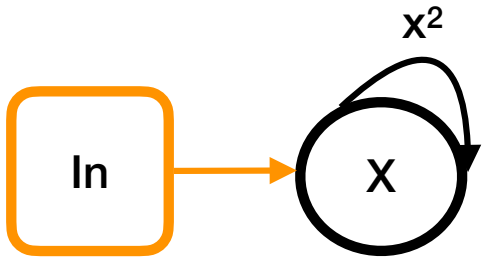
$$0 = 2x \pm 1, x = \pm.5$$

Neuro & Biomorphic Engineering LAB

The Neural Engineering Framework (NEF)

Dynamics





$$\dot{x} = \frac{1}{\tau}(x^2 - x)$$

Input = 0.2: $0 = x^2 - x + .2 = (x - .72)(x - .27)$

Input = 0.4: Solution is a complex number

Input = 0. : x = 0 , 1

The Neural Engineering Framework (NEF)

Dynamics

Synthesis

- What if there's some differential equation we really want to implement?
  - We want: $\dot{x} = f(x)$
  - So we do a recurrent connection of $f'(x) = \tau f(x) + x$
  - The resulting model will end up implementing $\dot{x} = \frac{1}{\tau}(\tau f(x) + x - x) = f(x)$
- What happens if there's an input as well?
- We'll call the input u from another population, which computes some function g(u)

$$x(t) = f(x(t)) * h(t) + g(u(t)) * h(t)$$

- Follow the same derivation steps to get:

$$\dot{x} = \frac{1}{\tau}(f(x) - x + g(u))$$

- So if you have some input that you want added to x, you need to scale it by τ
- This lets us do any differential equation of the form: $\dot{x} = f(x) + g(u)$

The Neural Engineering Framework (NEF)

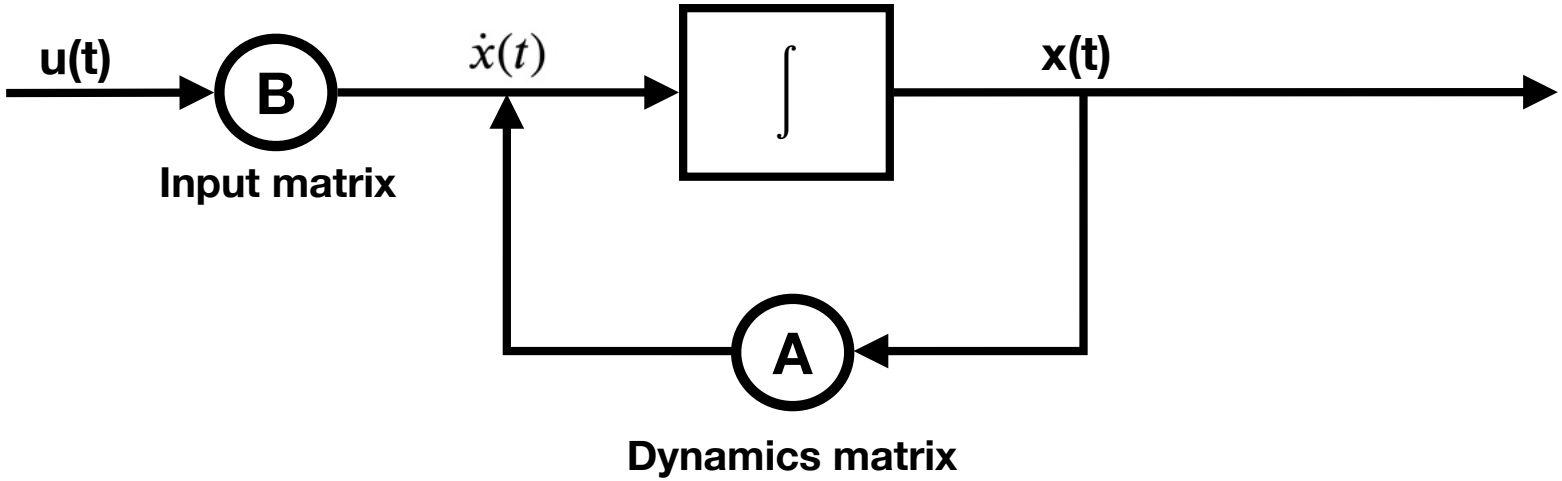
Dynamics

Let's consider a linear system

$$\dot{x}(t) = Ax(t) + Bu(t)$$

State vector
Summerize the effect of all past input

Input
The control vector



The Neural Engineering Framework (NEF)

Dynamics

Let's consider a linear system

$$\dot{x}(t) = Ax(t) + Bu(t)$$

- Laplace Transform: $sX(s) = AX(s) + BU(s)$ (standard equations)
- Laplace on the neural control: $X(s) = \frac{1}{1+s\tau}(A'X(s) + B'U(s))$

H(s)

$$X(s) + \tau sX(s) = (A'X(s) + B'U(s))$$

$$sX(s) = \frac{1}{\tau}(A'X(s) + B'U(s) - X(s))$$

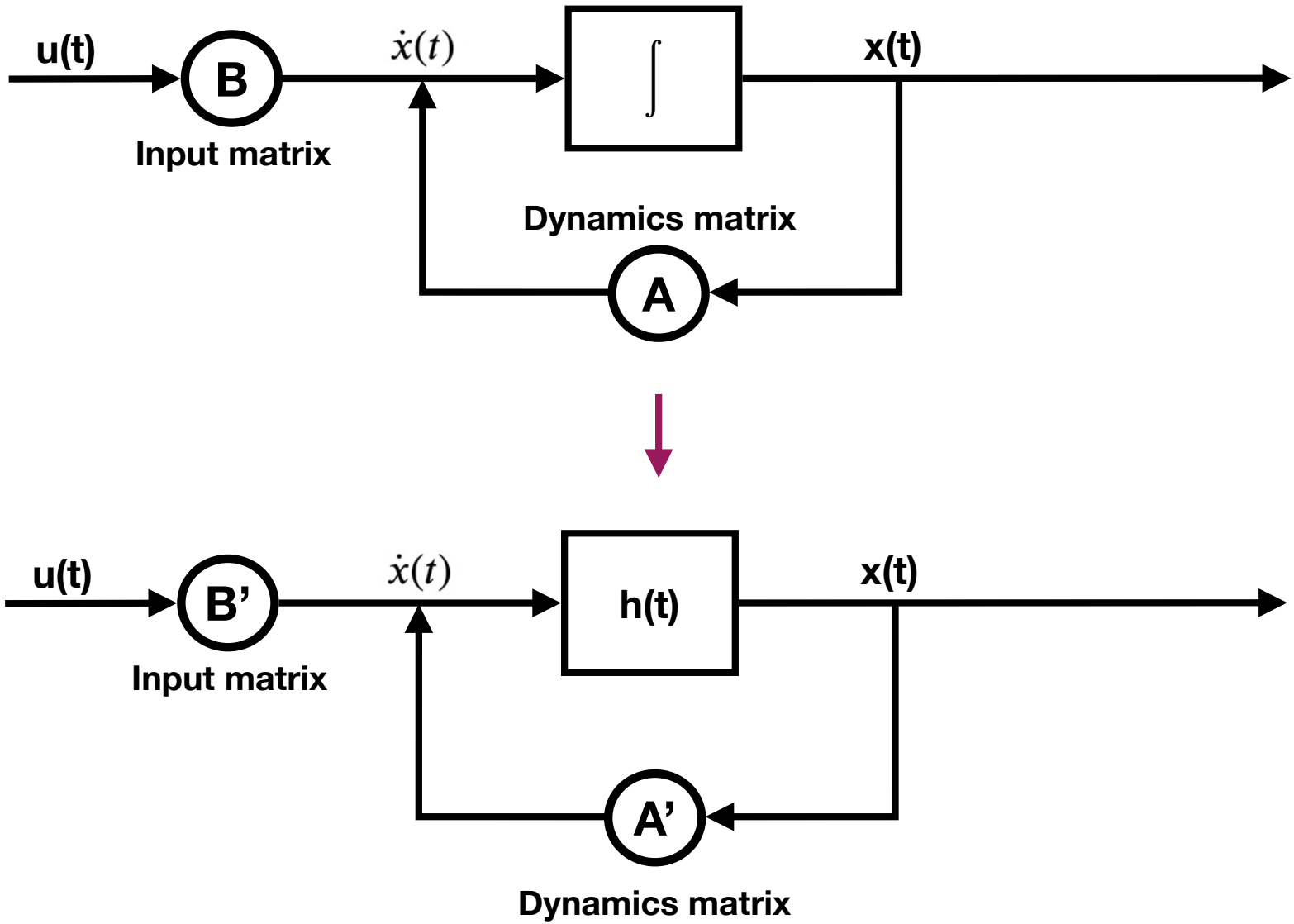
$$sX(s) = \frac{1}{\tau}((A' - I)X(s) + B'U(s))$$

Identity matrix

- Making the 'standard' and 'neural' equations equal to one another, we find that for any system with a given A and B, the A' and B' of the equivalent neural system are given by: $A' = \tau A + I$ and $B' = \tau B$

The Neural Engineering Framework (NEF)

Dynamics



The Neural Engineering Framework (NEF)

Dynamics

Let's consider a nonlinear system

In fact, these same steps can be taken to account for nonlinear control systems as well.

Outside of the scope for this course…

The Neural Engineering Framework (NEF)

Dynamics

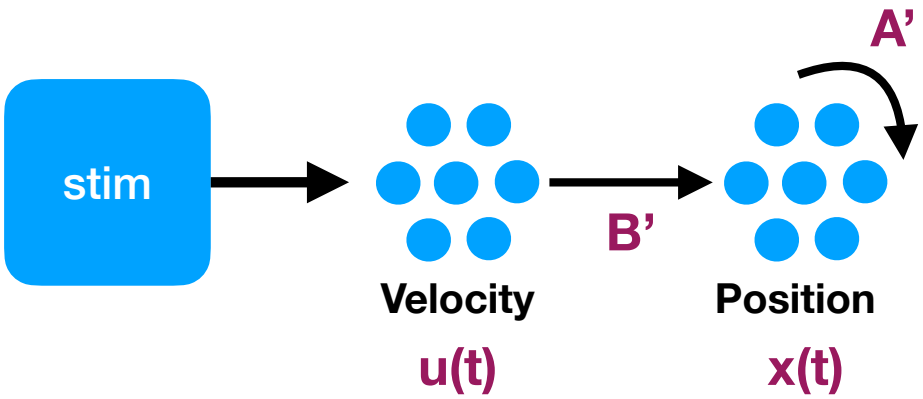
Example: Eye Control

- Part of the brainstem called the nuclei prepositus hypoglossi
- Input is eye velocity v
- Output is eye position x
- This is an integrator (x is the integral of v): $\dot{x} = v$
- It's a linear system, so, to get it in the standard control form: $\dot{x} = Ax + Bu$
  A=0 and B=1
- We have: $A' = \tau 0 + I = 1$ and $B' = \tau 1 = \tau$

The Neural Engineering Framework (NEF)

Dynamics

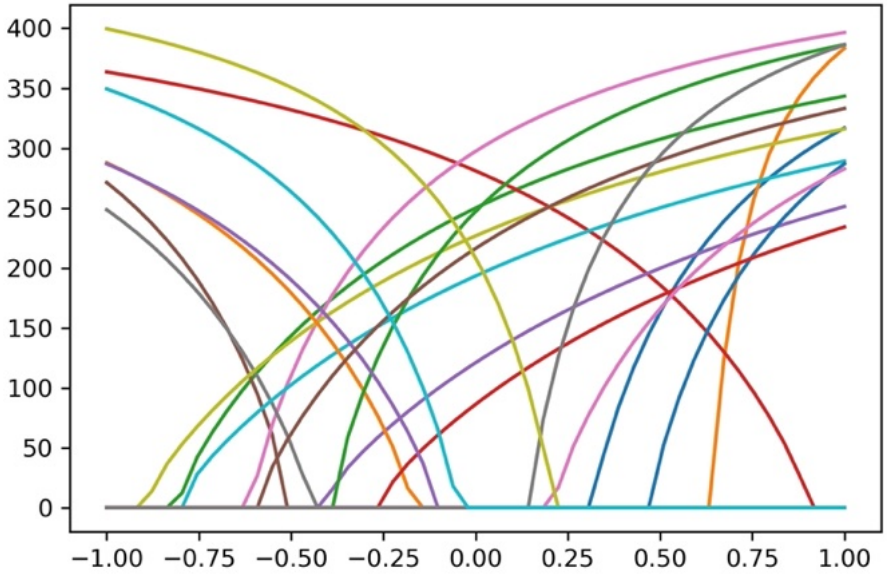
```
stim = nengo.Node(piecewise({.3:1, .6:0 }))
velocity = nengo.Ensemble(100, dimensions=1)
position = nengo.Ensemble(20, dimensions=1)

def feedback(x):
    return 1*x

conn = nengo.Connection(stim, velocity)
conn = nengo.Connection(velocity, position, transform=tau, synapse=tau)
conn = nengo.Connection(position, position, function=feedback, synapse=tau)
```

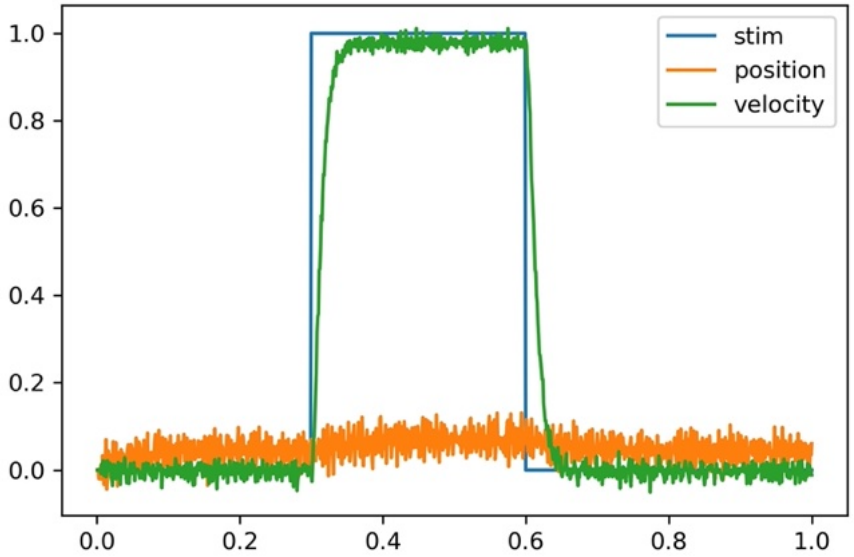


The Neural Engineering Framework (NEF)

Dynamics

Example: Eye Control



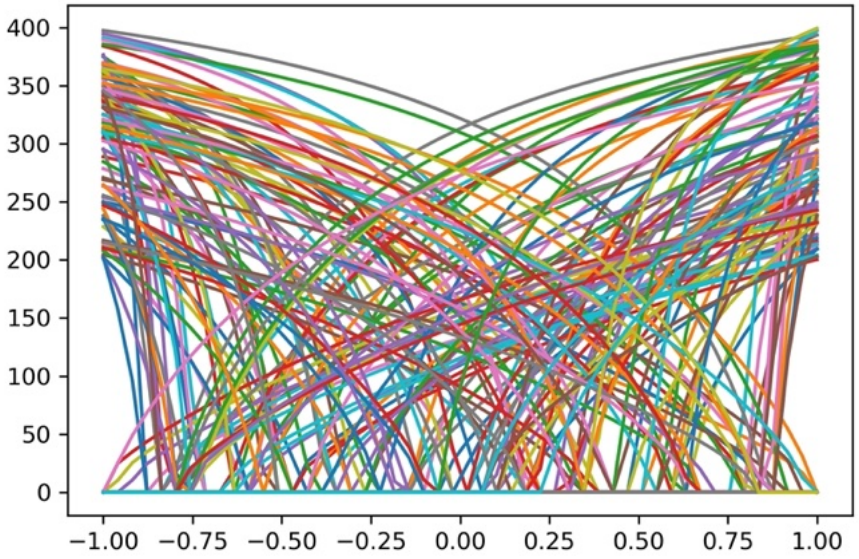


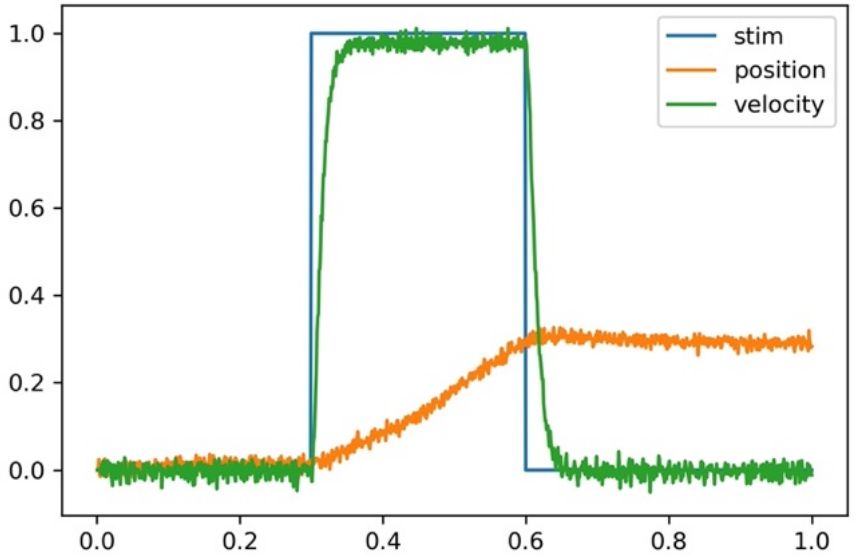
Not so good…

The Neural Engineering Framework (NEF)

Dynamics

Example: Eye Control





More neurons…

Neuro & Biomorphic Engineering LAB

The Neural Engineering Framework (NEF)

Dynamics

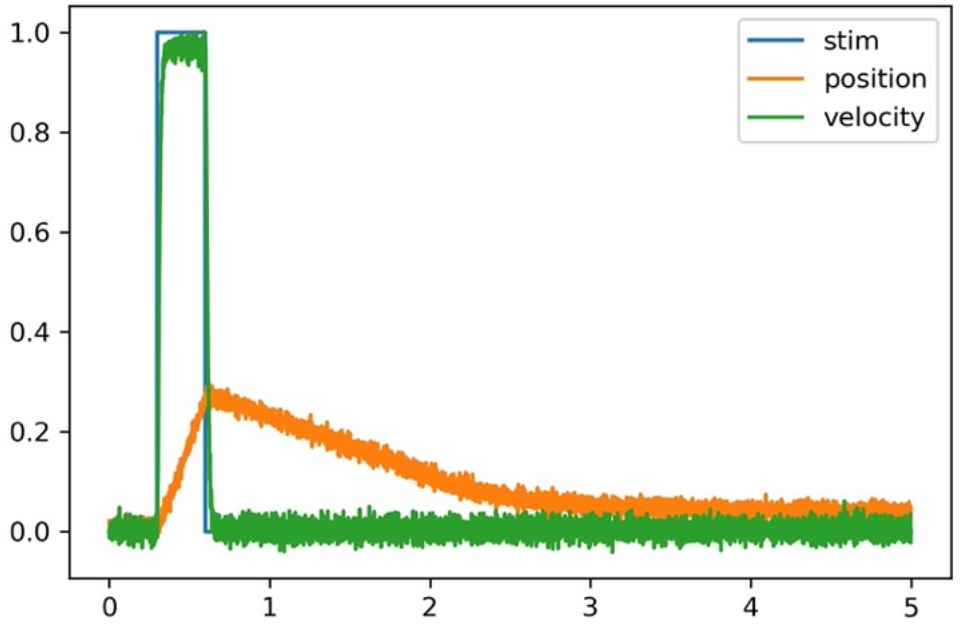
Example: Eye Control

- But real eyes aren't perfect integrators
- If you get someone to look at something, then turn off the lights but tell them to keep looking in the same direction, their eye will drift back to centre (with about 70s time constant)

$$\dot{x} = -\frac{1}{\tau_c}x + v$$

$$A' = \tau\frac{-1}{\tau_c} + 1$$

$$B' = \tau$$

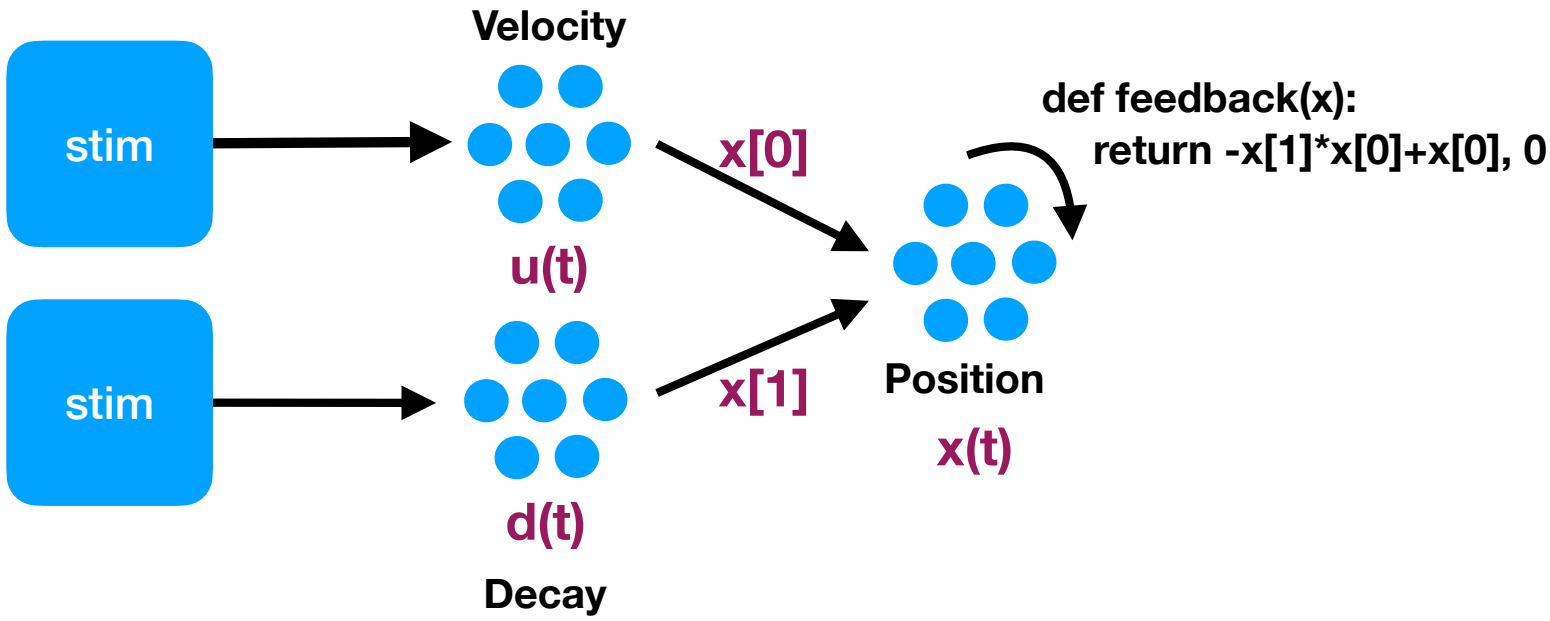


The Neural Engineering Framework (NEF)

Dynamics

Example: Controlled integrator

- What if we want an integrator where we can adjust the decay on-the-fly?
- Separate input telling us what the decay constant d should be: $\dot{x} = -dx + v$
- Going to 2D neuron ensembles!

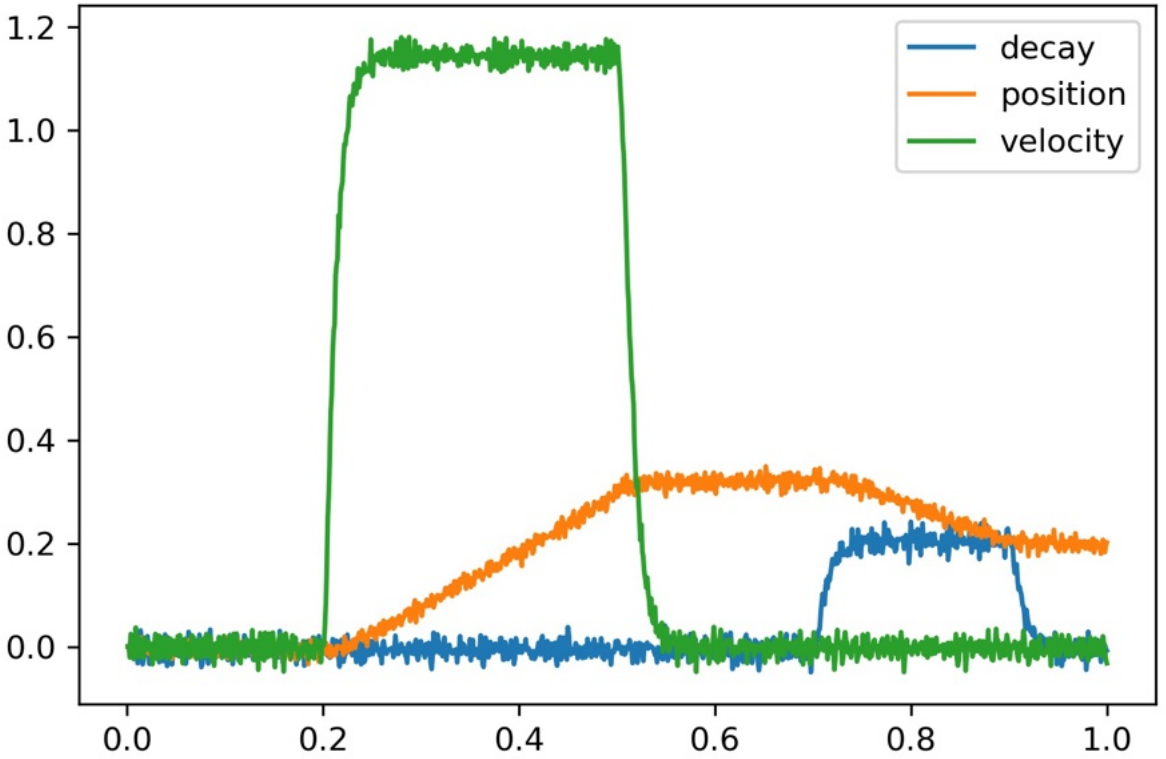


The Neural Engineering Framework (NEF)

Dynamics

Example: Controlled integrator

- What if we want an integrator where we can adjust the decay on-the-fly?
- Separate input telling us what the decay constant d should be: $\dot{x} = -dx + v$
- Going to 2D neuron ensembles!



The Neural Engineering Framework (NEF)

Dynamics

Example: Oscillator

- A 2D oscillator, which alternates the values of x_0 and x_1, at a rate r, can be defined recurrently using:

$$F = -kx = m\ddot{x} \text{ let } \omega = \sqrt{\frac{k}{m}}$$

$$\frac{d}{dt}\begin{bmatrix}\omega x \\ \dot{x}\end{bmatrix} = \begin{bmatrix}0 & \omega \\ -\omega & 0\end{bmatrix}\begin{bmatrix}x_0 \\ x_1\end{bmatrix} \longrightarrow \dot{x} = [x_1, -x_0] \xrightarrow{\text{WHY?}} \begin{pmatrix}x_1 \\ x_2\end{pmatrix} = \begin{pmatrix}1 & r \\ -r & 1\end{pmatrix}\begin{pmatrix}x_1 \\ x_2\end{pmatrix}$$

```
stim = nengo.Node(lambda t: [.5,.5] if t<.02 else [0,0])

osc = nengo.Ensemble(200, dimensions=2)

def feedback(x):
    return x[0]+freq*x[1], -freq*x[0]+x[1]

nengo.Connection(osc, osc, function=feedback, synapse=.01)
nengo.Connection(stim, osc)
```

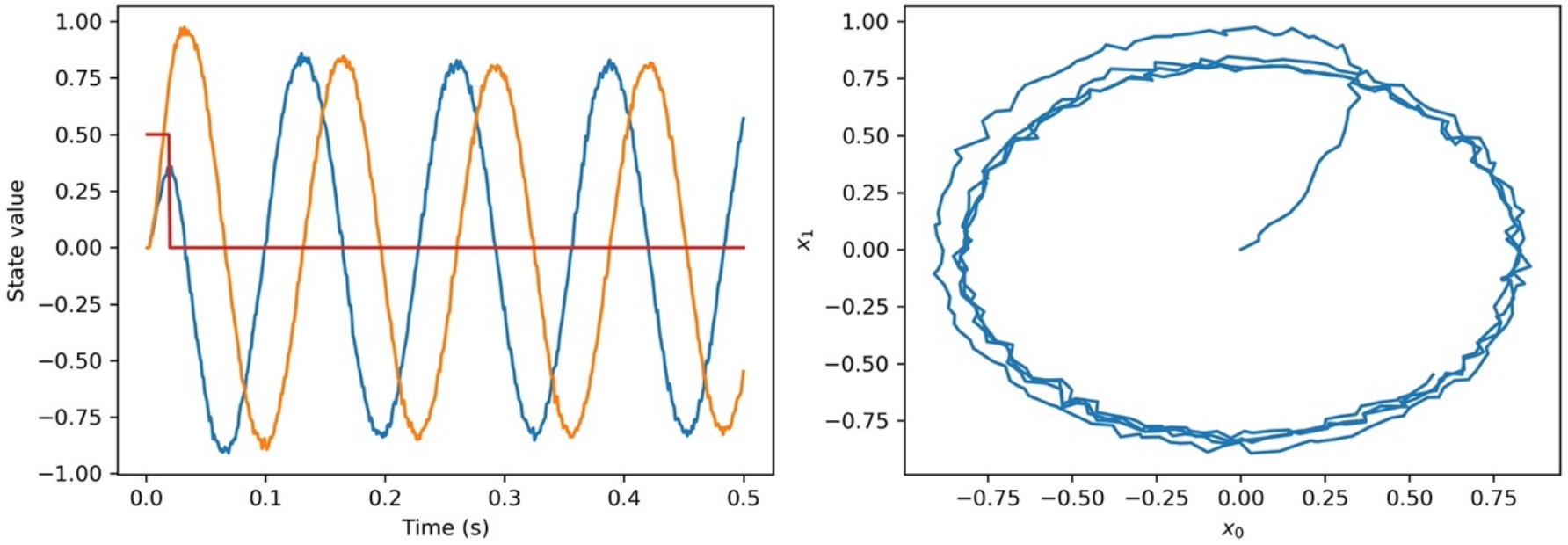
The Neural Engineering Framework (NEF)

Dynamics

Example: Oscillator

- A 2D oscillator, which alternates the values of x_0 and x_1, at a rate r, can be defined recurrently using:

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 & r \\ -r & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = Ax$$



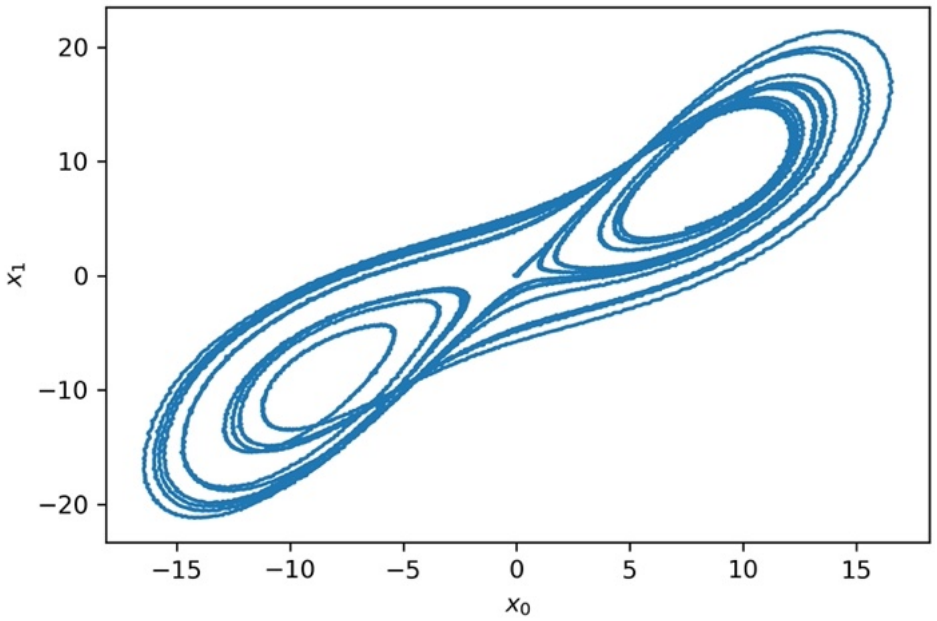
The Neural Engineering Framework (NEF)

Dynamics

Example: Lorenz Attractor

$$\dot{x} = [10x_1 - 10x_0, -x_0x_2 - x_1, x_0x_1 - \frac{8}{3}(x_2 + 28) - 28]$$

```
x = nengo.Ensemble(n_neurons=600, dimensions=3, radius=30)

    synapse = 0.1
    def lorenz(x):
        sigma = 10
        beta = 8.0/3
        rho = 28

        dx0 = -sigma * x[0] + sigma * x[1]
        dx1 = -x[0] * x[2] - x[1]
        dx2 = x[0] * x[1] - beta * (x[2] + rho) - rho

        return [dx0 * synapse + x[0],
                dx1 * synapse + x[1],
                dx2 * synapse + x[2]]

    nengo.Connection(x, x, synapse=synapse, function=lorenz)
```



The Neural Engineering Framework (NEF)

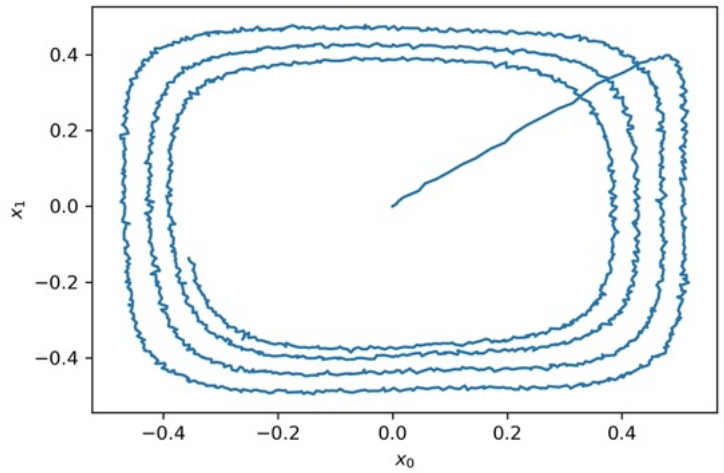
Dynamics

Example: Arbitrary Oscillator

- Since we can implement any function, we're not limited to linear oscillators
- What about a "square" oscillator?

$$\dot{x} = \begin{cases} [r, 0] & \text{if } |x_1| > |x_0| \wedge x_1 > 0 \\ [-r, 0] & \text{if } |x_1| > |x_0| \wedge x_1 < 0 \\ [0, -r] & \text{if } |x_1| < |x_0| \wedge x_0 > 0 \\ [0, r] & \text{if } |x_1| < |x_0| \wedge x_0 < 0 \end{cases}$$

```
tau = 0.02
r=6

def feedback(x):
    if abs(x[1])>abs(x[0]):
        if x[1]>0: dx=[r, 0]
        else: dx=[-r, 0]
    else:
        if x[0]>0: dx=[0, -r]
        else: dx=[0, r]
    return [tau*dx[0]+x[0], tau*dx[1]+x[1]]

with model:
    stim = nengo.Node(lambda t: [.5,.5] if t<.02 else [0,0])
    square_osc = nengo.Ensemble(1000, dimensions=2)
    nengo.Connection(square_osc, square_osc, function=feedback, synapse=tau)
    nengo.Connection(stim, square_osc)
```



How to build a BRAIN?

Symbols

Memory

Action Selection

Spatial Representation

The Neural Engineering Framework (NEF)

Symbols

- We've seen how to represent vectors in neurons
- And how to compute functions on those vectors
- And dynamical systems
- But how can we do anything like human language?
- How could we represent the fact that "the number after 8 is 9“
    - Or "dogs chase cats“
    - Or "Anne knows that Bill thinks that Charlie likes Dave“
- Does the NEF help us at all with this?
    - Or is this just too hard a problem yet?