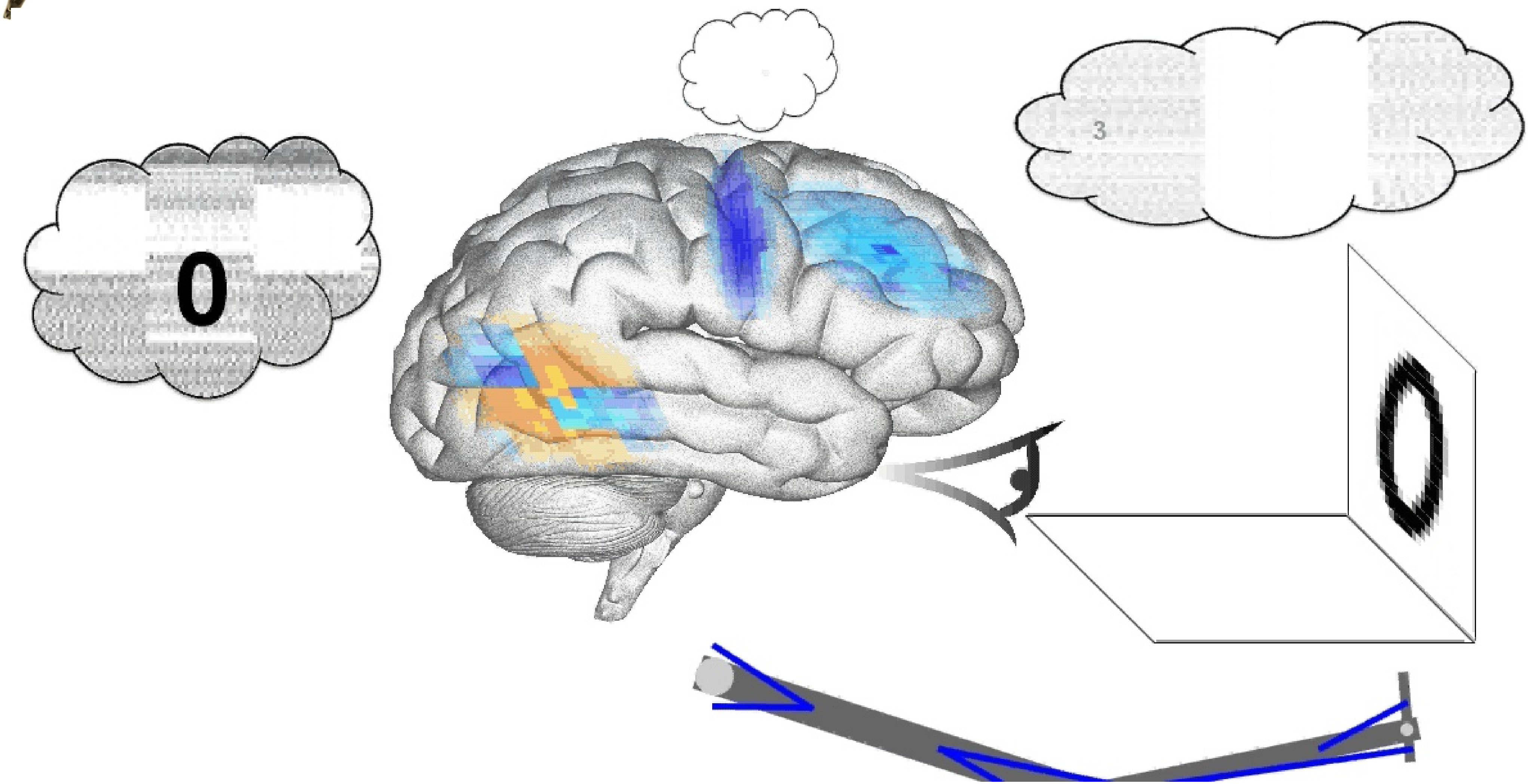
Computing with neurons

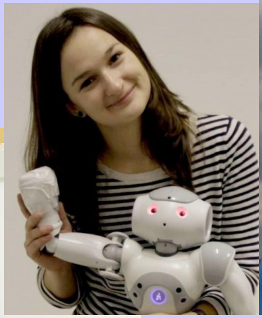
Session 1: Neural Engineering



Terrence C. Stewart, Centre for Theoretical Neuroscience, University of Waterloo

Centre for Theoretical Neuroscience University of Waterloo

- Chris Eliasmith



CTN and ABR

Understanding the mind

- What are the algorithms underlying cognition?
  - What is the mind doing?
- How can we know what the mind is doing?
  - How can we test these theories?

Understanding the mind

- Build computer models of the mind
- Mechanistic models
  - Internal components that map onto real system
- Why do we need a computational model?
  - Not analytically tractable

Cognitive Modelling

- Choose a phenomenon
  - Examine human behaviour
  - Build a computer program
  - Compare behaviours of program and human
- Many domains
  - Memory, mental arithmetic, reward learning
- Problem
  - How do we know if we're right?

Cognitive Architectures

- More constrained approach
  - Define a bunch of basic modules
    - Declarative memory
    - Visual recognition
    - Hand movement
    - Procedural memory
  - Use the same set of modules to do many tasks
    - It's not like we suddenly get new brain areas for each new task

Cognitive Architectures

- ACT-R
  - Declarative memory

$$B_i = \ln\left(\sum_{j=1}^{n} t_j^{-d}\right) + \beta_i$$

  - Procedural memory
    - IF-THEN rules
      - IF I'm counting and I'm at THREE then go to FOUR
  - Parameter values
    - d = 0.5
    - 50 ms per rule
      - Found by looking at human data across many conditions
    - What are the limits on the procedural rules?

Cognitive Architectures

- Wide variety of tasks
  - Mental arithmetic
  - Estimating time
  - Visual search
  - Air-traffic control
  - Military squad co-ordination
  - Language interpretation
  - Driving a car
  - Dialing a phone number
  - Driving a car while dialing a phone number

Cognitive Architectures

- How does this help?
  - More constraints
- Same components do many different tasks
- Parameter values shouldn't change (much)
  - Or theory can say when they change
- Predicting many different aspects of behaviour with a small set of components

What about the brain?

- Should we pay attention to it?
- Why would it matter for algorithms?
  - Why not just look for the best algorithm?
  - Why constrain ourselves?

Advantage 1
More predictions

- A brain-based model will predict more than just overt behaviour
  - Connectivity
  - Firing patterns
  - Results of lesions
  - Timing
  - Effects of drugs

Advantage 2
Different algorithms

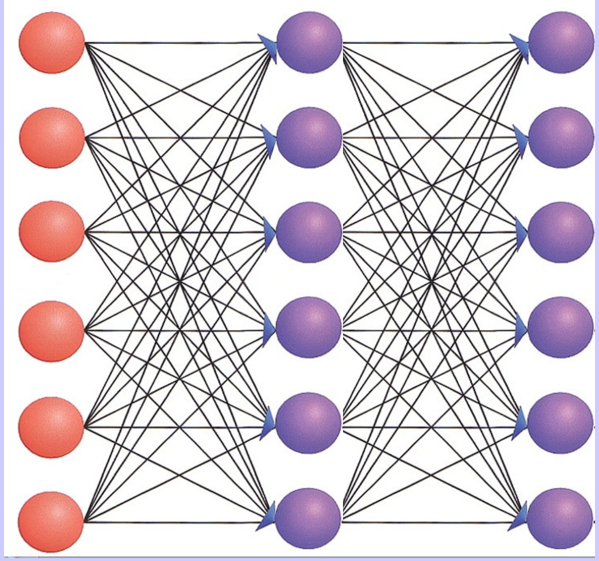
- Infinite numbers of algorithms to consider
- We implement algorithms on computers
  - So we are biased toward considering algorithms *that are easy to program*
- Instead, let's determine the types of algorithms that neurons would be good at implementing
  - Then make software tools to make those types of algorithms easy to program

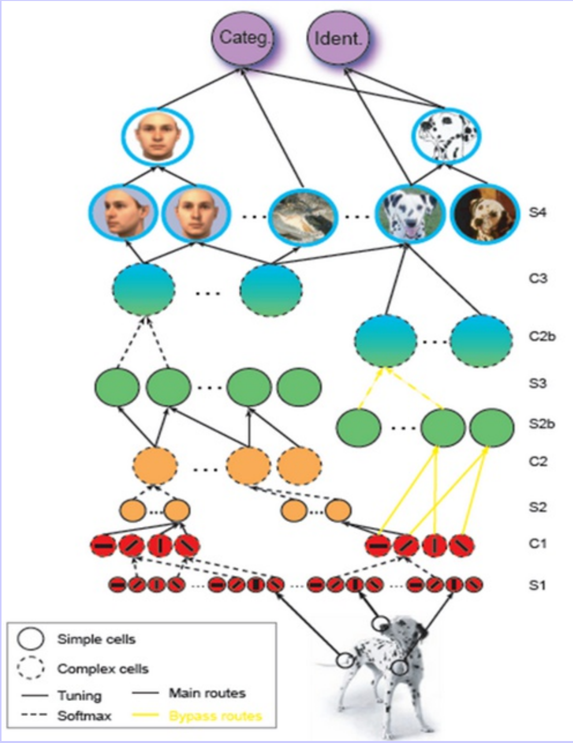
The Brain

- What is the brain?
- How should we think about the brain?
  - 140,000,000,000,000,000,000,000,000 atoms?
  - 100,000,000,000 neurons?
  - 52 brain areas?

Connectionism

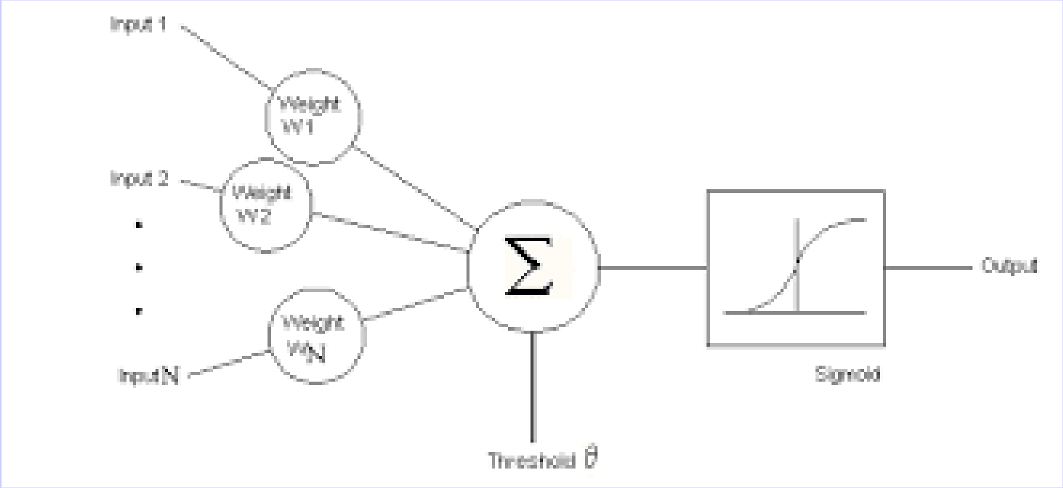
- Neural networks
  - Many components
  - Many connections
  - Components add their inputs, perform some non-linearity to produce outputs





Connectionism

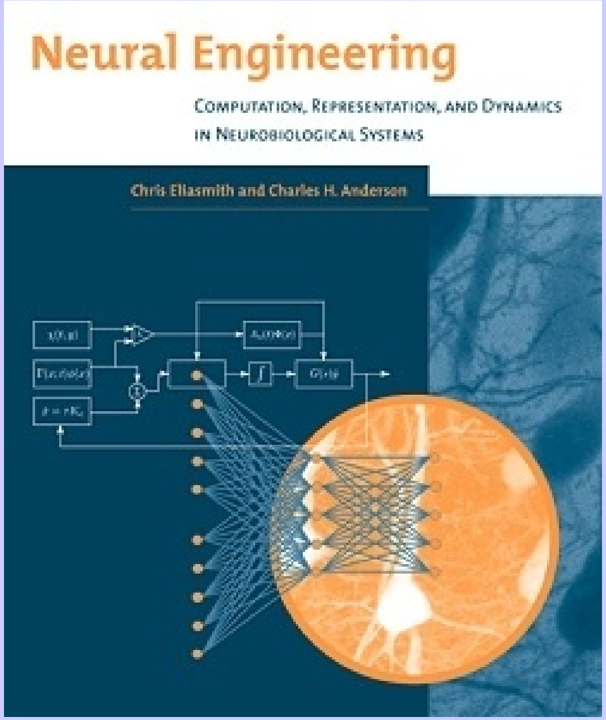
- How do we decide what the components can do?
  - Common choice: sigmoid neuron
  - Why that one?
    - Easy to program



- How do we get connection weights?
  - Start random, apply learning rule
  - Gets better and better at task (maybe)
    - Lots of computing needed

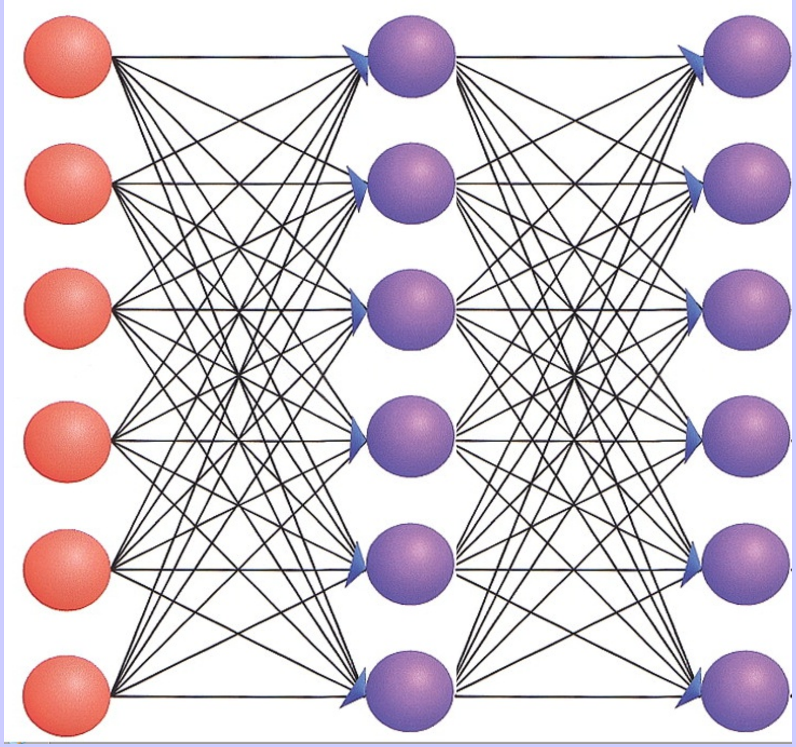
Neural Engineering Framework

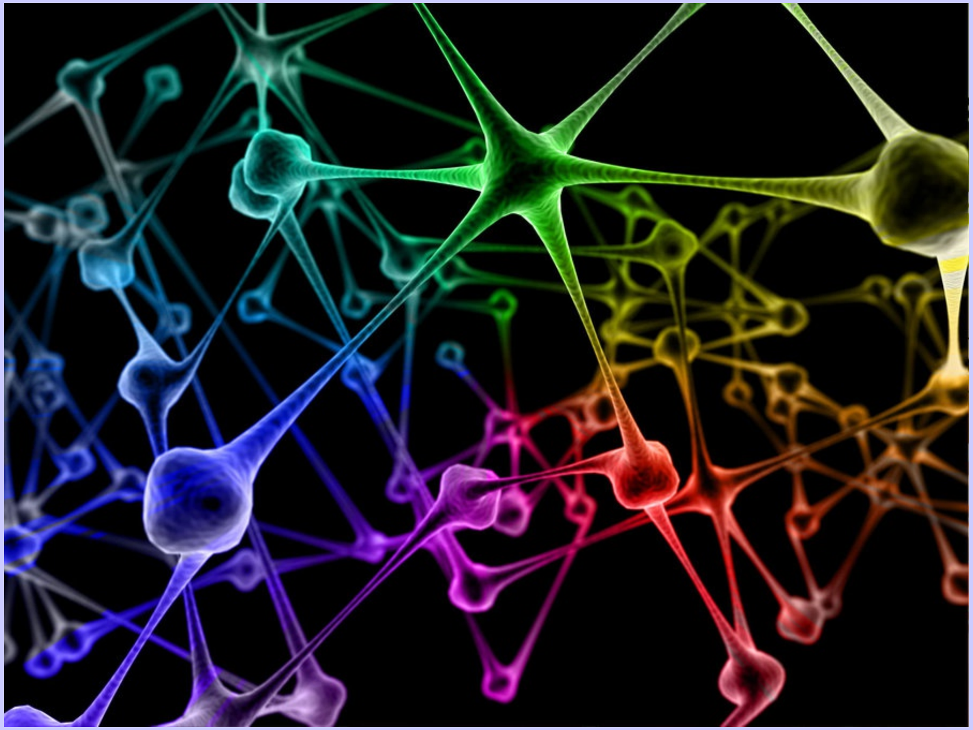
- (Eliasmith & Anderson, 2003)
- Is there another way?
  - What are realistic neurons good at computing?
  - Can this help resolve the connection weight problem?



A neuron

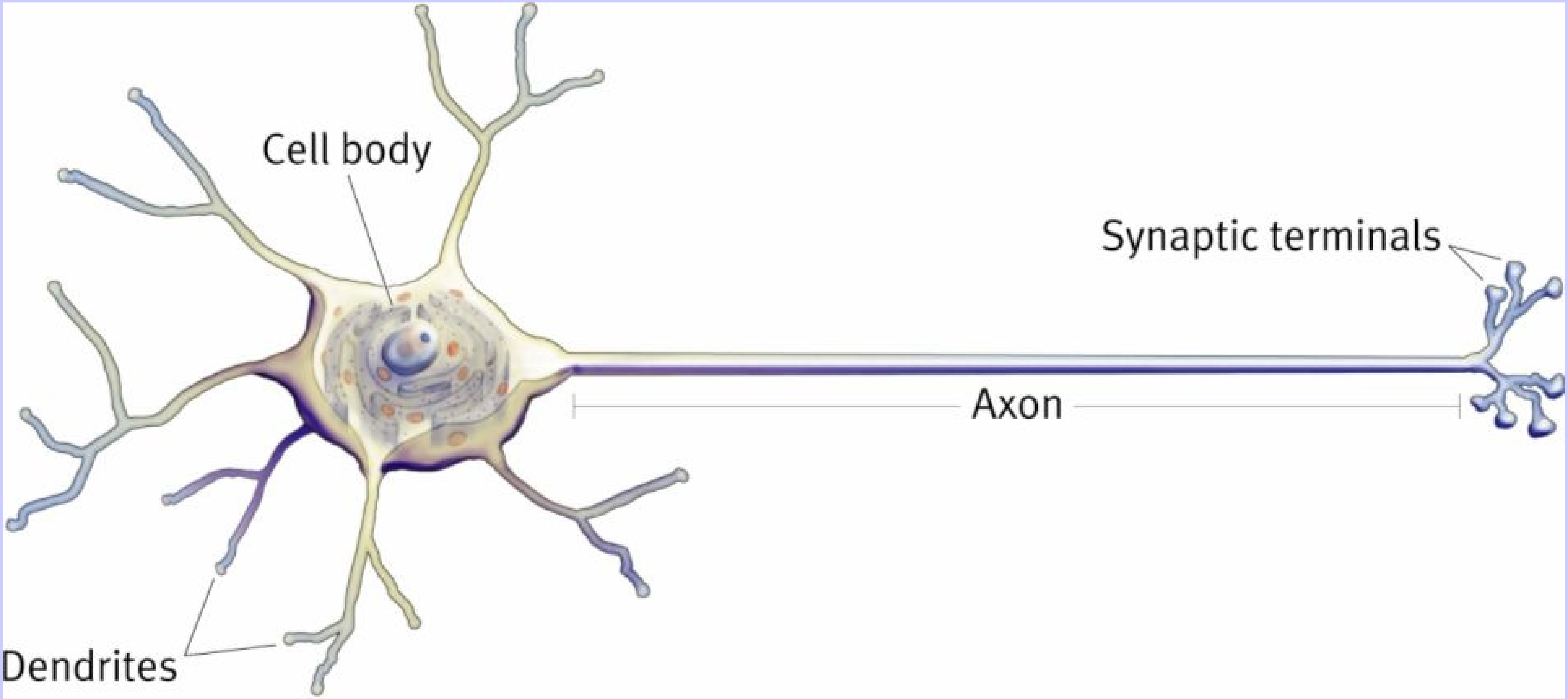
- What is a neuron really like?





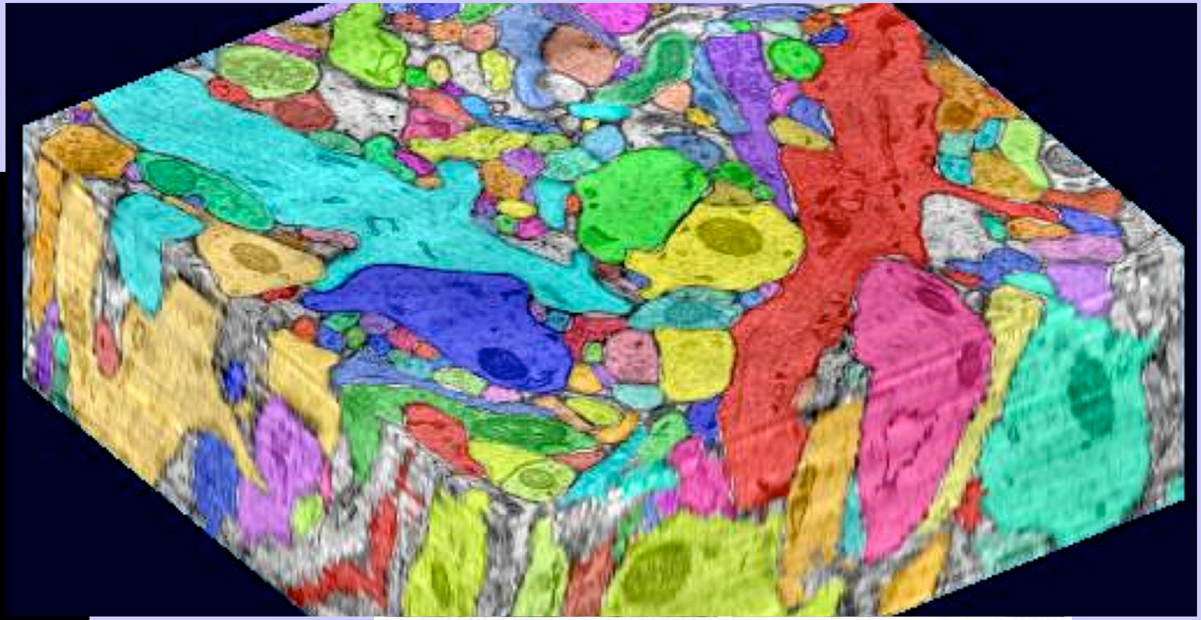
A neuron

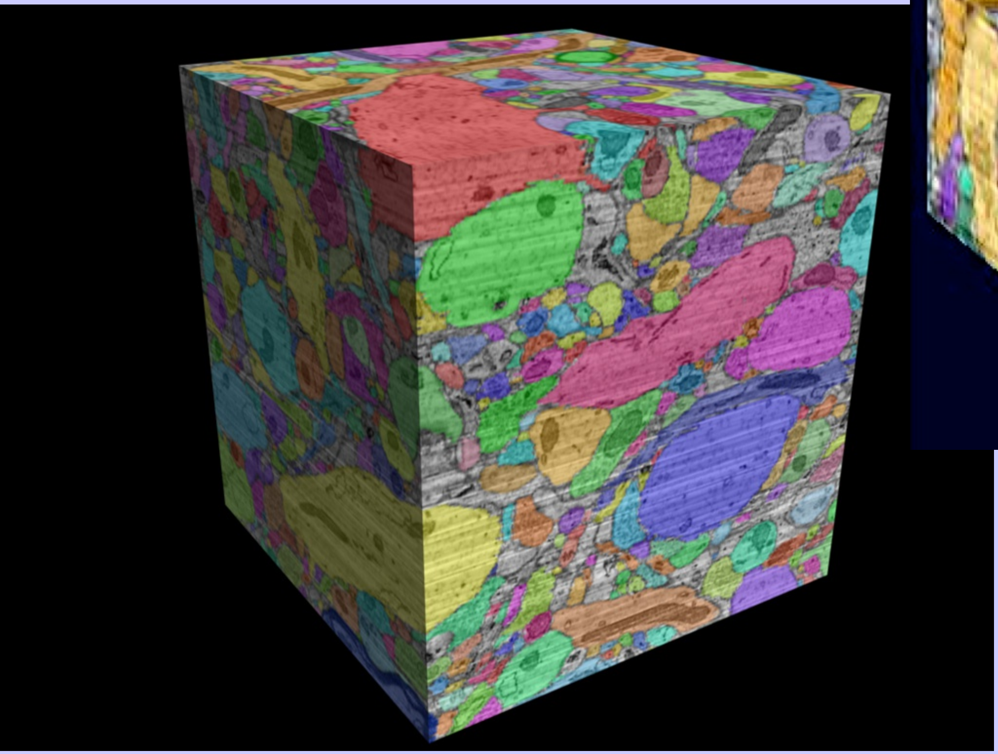
- What is a neuron really like?

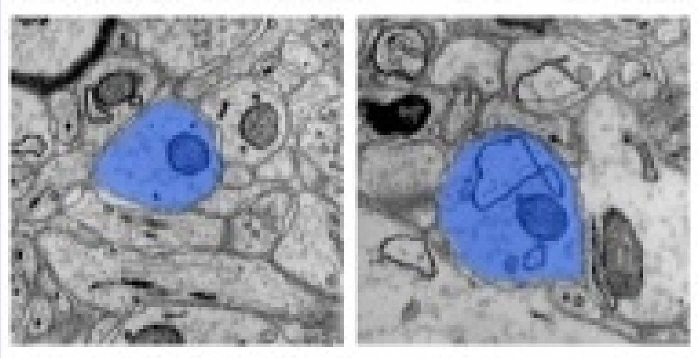


A neuron

- What is a neuron really like?

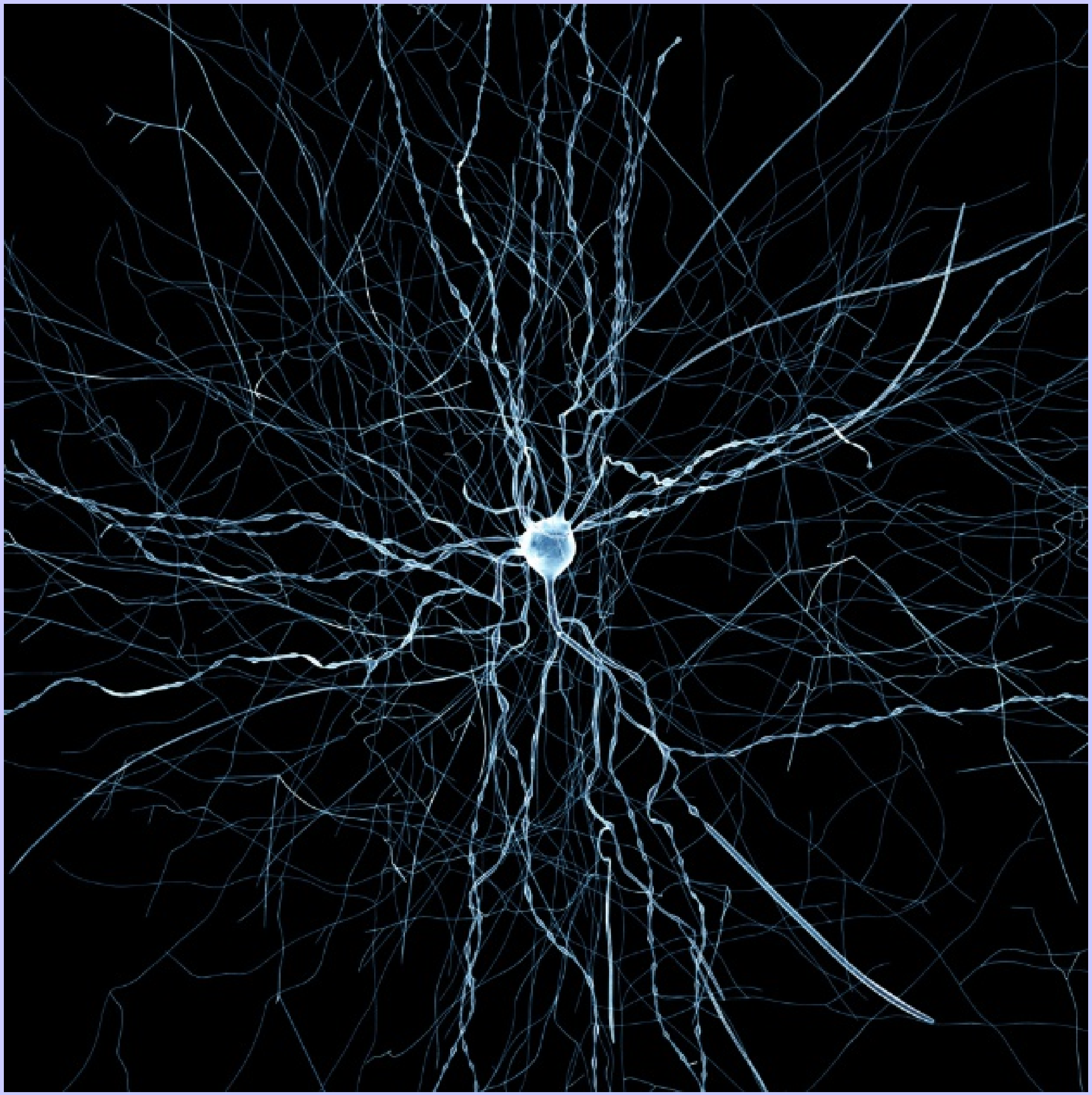


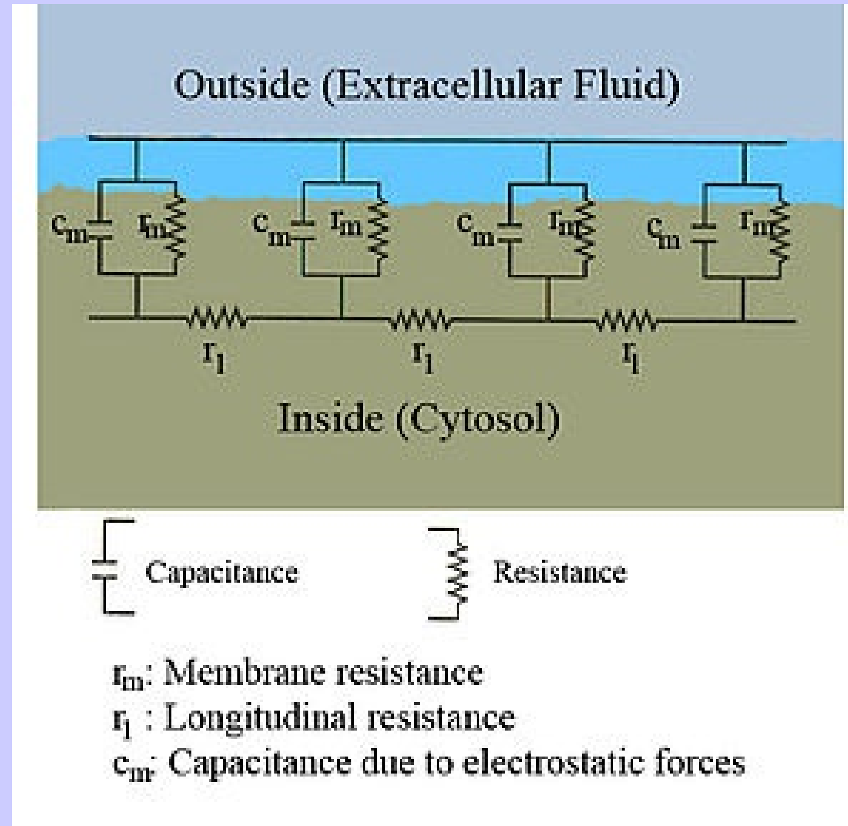




Link:crumb of mouse brain

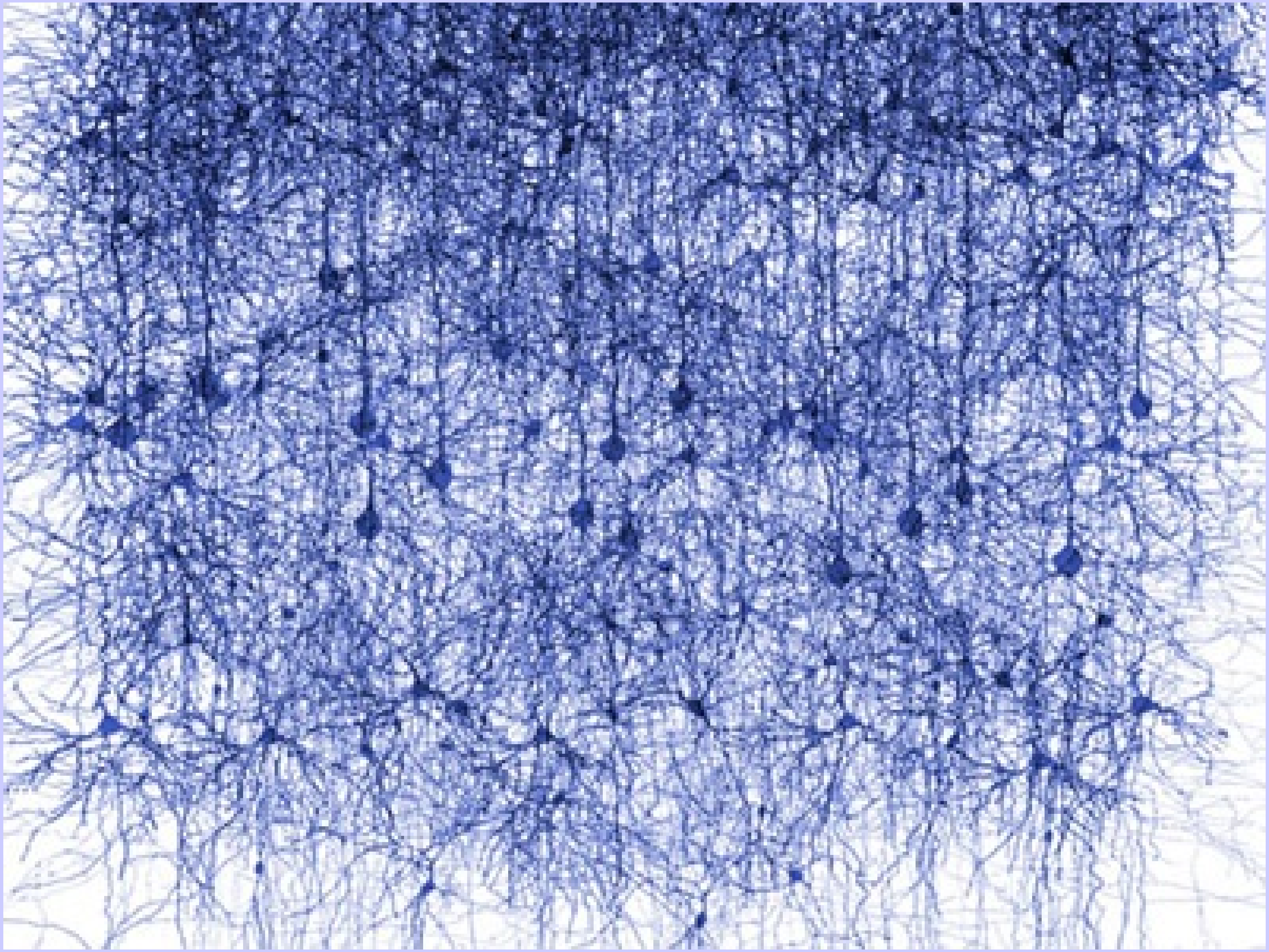
A neuron



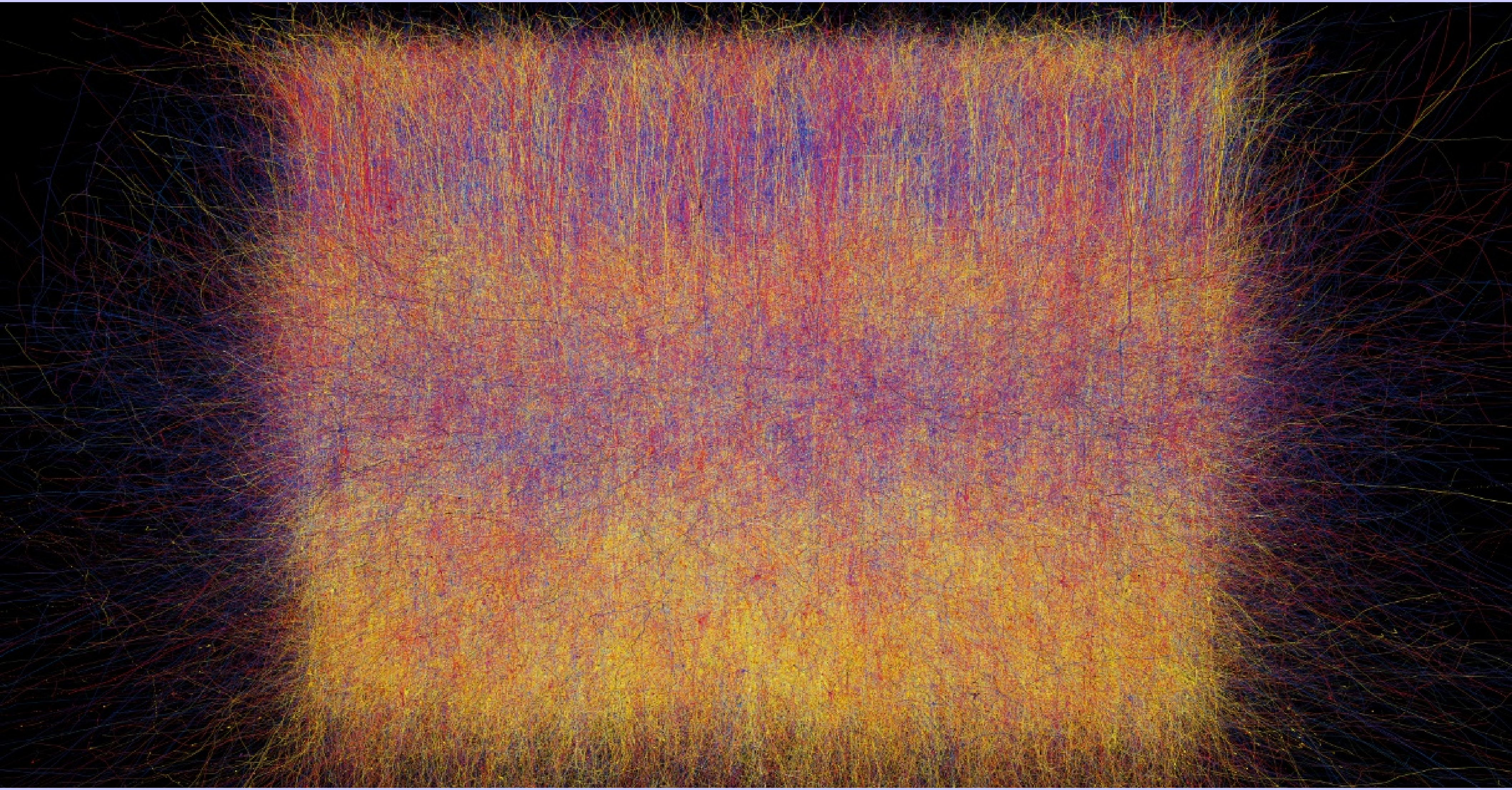


$$\frac{r_m}{r_l}\frac{\partial^2 V}{\partial x^2} = c_m r_m \frac{\partial V}{\partial t} + V$$

Many neurons
(and synapses)



Many many neurons
(and synapses)



(Markram et al., Cell, 2015)

31,000 neurons; 37 million synapses

How much detail?

- Do we need all that complexity?
  - How do we know when to stop?
- How do other sciences deal with this?

How much detail?

- Do we need all that complexity?
  - How do we know when to stop?
- How do other sciences deal with this?
  - Physics (gravity): sometimes Newton is enough detail, sometimes you need Einstein
- The level of detail needed depends on the question being asked
  - e.g. drug effects may require a detailed model
  - But do we need it for understanding behaviour?

Behaving Systems

- Brains are for behaving
  - Sensory input, muscle outputs
- If we want computational neuroscience to explain what people do and how they do it, then the models need to produce that behaviour
  - Given similar inputs as the real system:
  - Produce similar outputs
- But behaviour requires many more neurons....

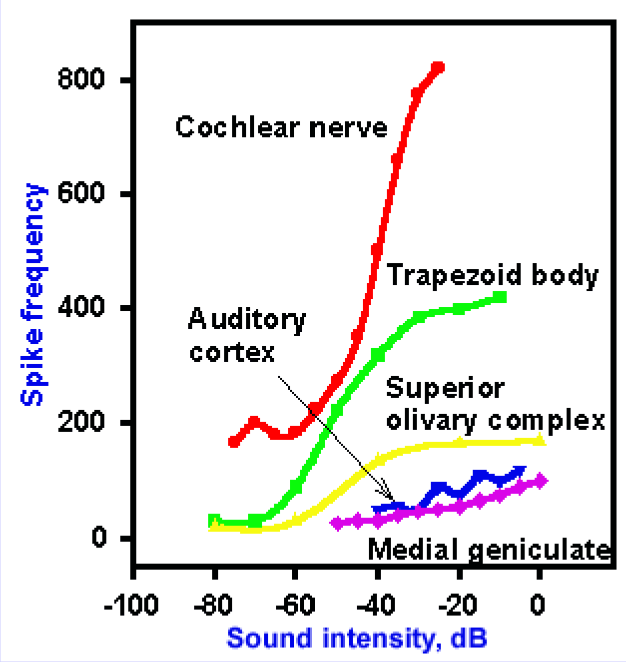
A neuron

- Could use a full supercomputer to simulate one neuron
- Have to make some abstraction
  - Start with something simple and uncontroversial
  - But everything we do could also be applied to more complex neurons

$$I(t) - \frac{V_{\mathrm{m}}(t)}{R_{\mathrm{m}}} = C_{\mathrm{m}} \frac{dV_{\mathrm{m}}(t)}{dt}$$

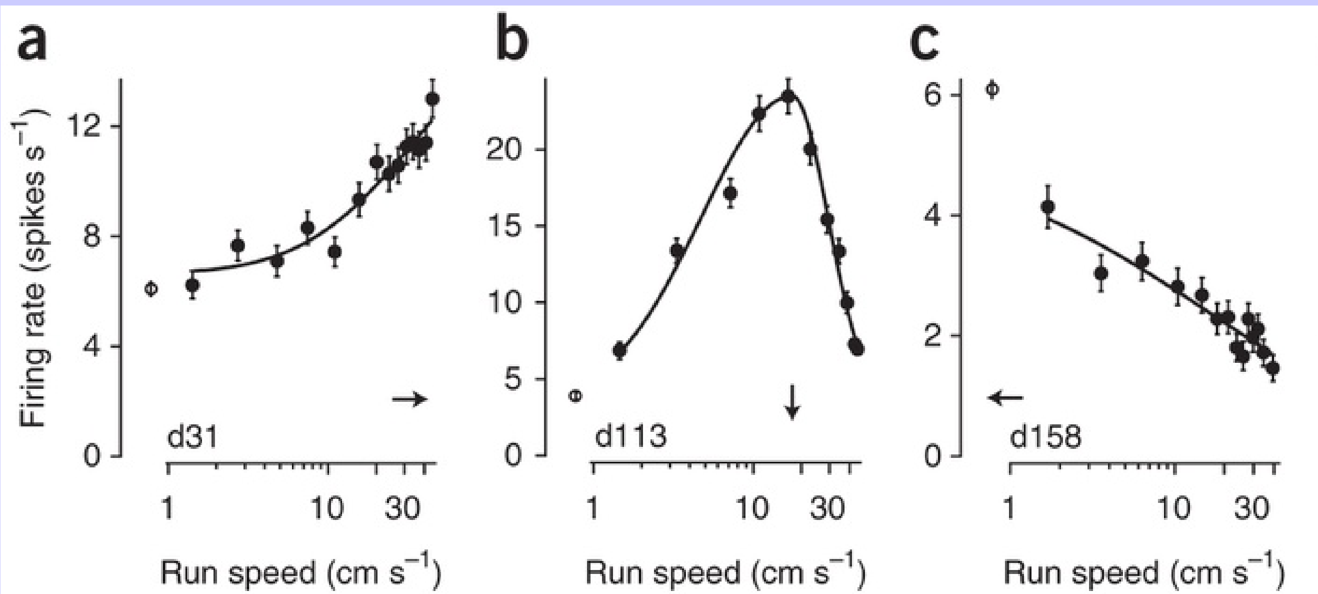
Tuning Curves

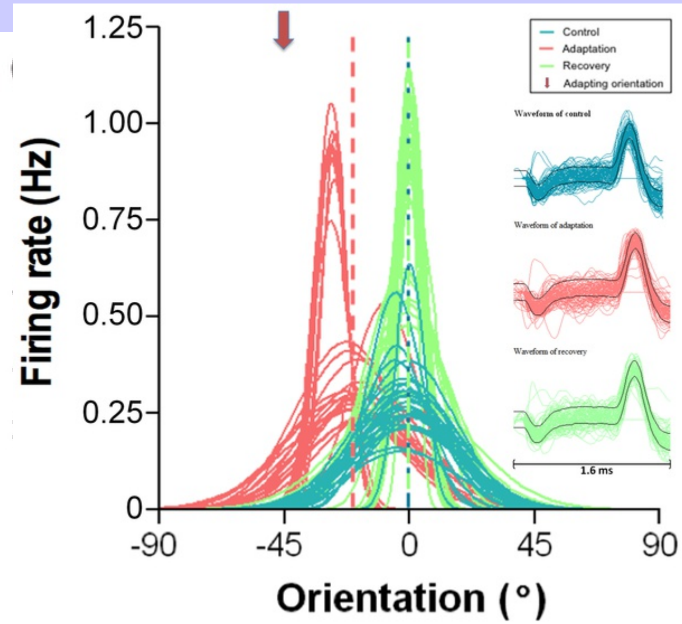
- How do neurons represent?
  - What is the relationship between the activity of a neuron *a* and the variable being represented *x*?
  - Sometimes this is easy:



Tuning Curves

- Other times, not so much





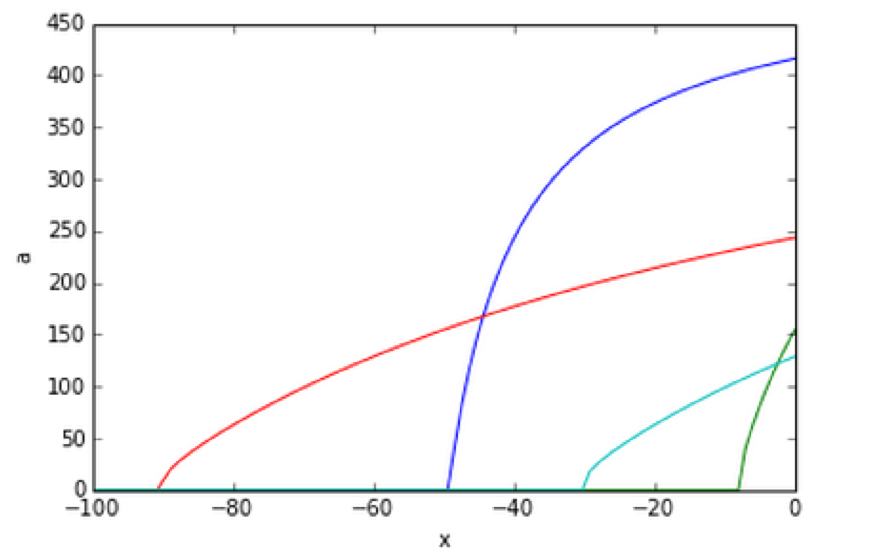
Tuning Curves

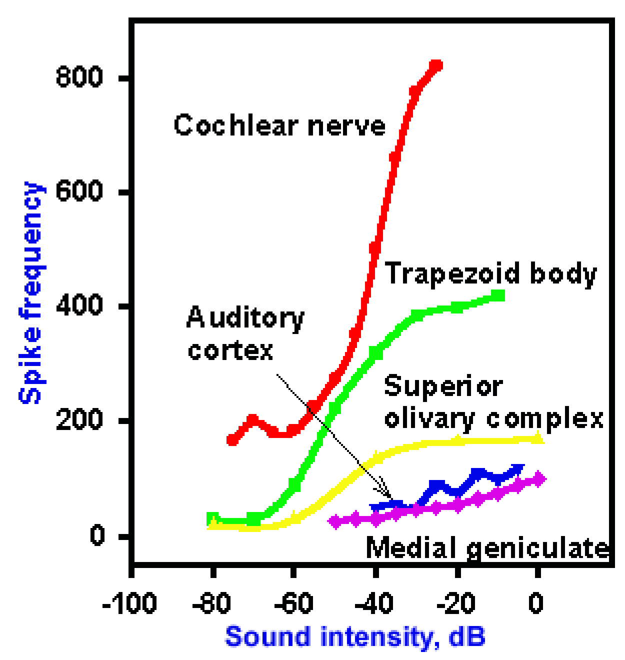
- Let's break these tuning curves down into two aspects
  - Mapping from *x* (the variable) to *J* (current)
    - This is about how this neuron's inputs are organized
  - Mapping from *J* (current) to *a* (activity)
    - This is about the intrinsic response
    - This can be as complex as you want (assuming you have the compute power to do so)
    - *a* can be spikes or rates
      - I'm going to plot it as rates for now

Tuning Curves

- Easy case:
  - $$J = \alpha x + J^{bias}$$
  - For activity, use standard LIF model for now

```
plot(x, n.rates(x, gain=1, bias=50), 'b') # x*1+50
plot(x, n.rates(x, gain=0.1, bias=10), 'r') # x*0.1+10
plot(x, n.rates(x, gain=0.5, bias=5), 'g') # x*0.05+5
plot(x, n.rates(x, gain=0.1, bias=4), 'c') #x*0.1+4))
```

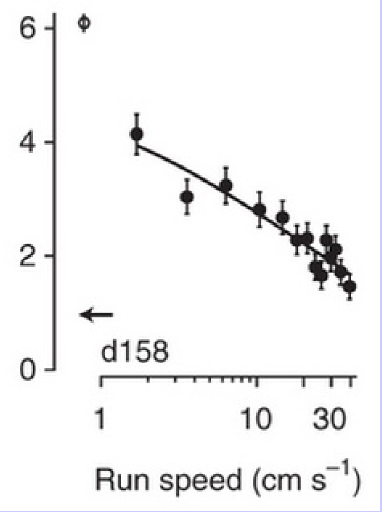




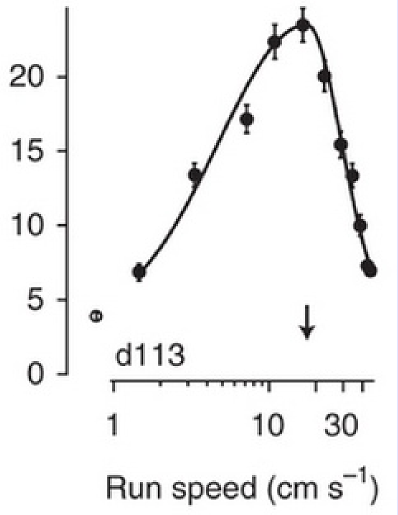
Tuning Curves

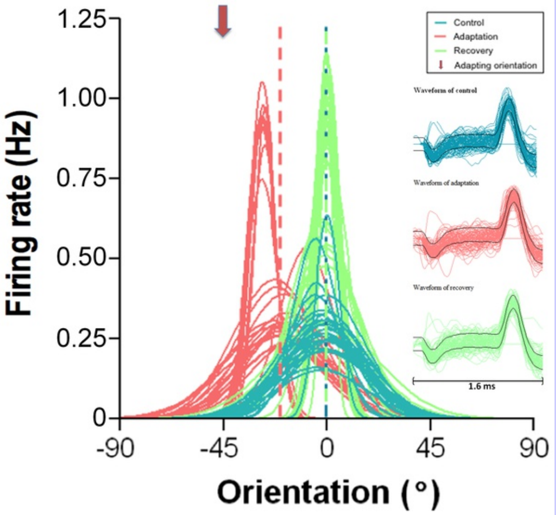
- What about these?
  - Mapping from x to J

$$J = -\alpha x + J^{bias}$$

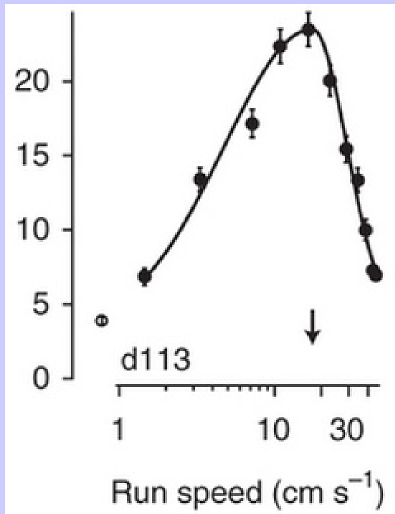


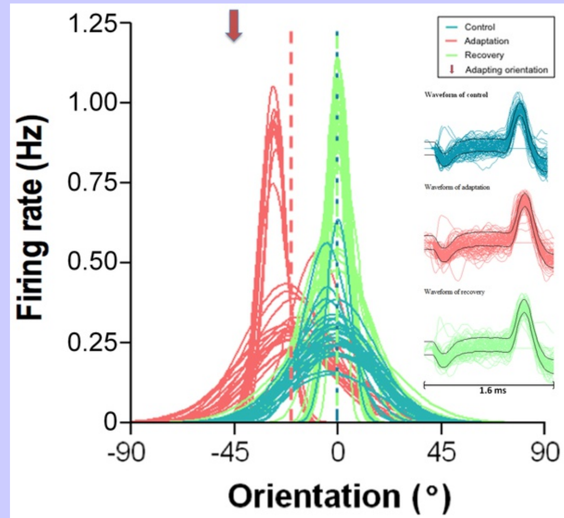
- Okay, what about these?





Tuning Curves





- 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?
- Let's think about x for a moment
  - x can be anything... scalar, vector, etc.
  - Does thinking of it as a vector help?

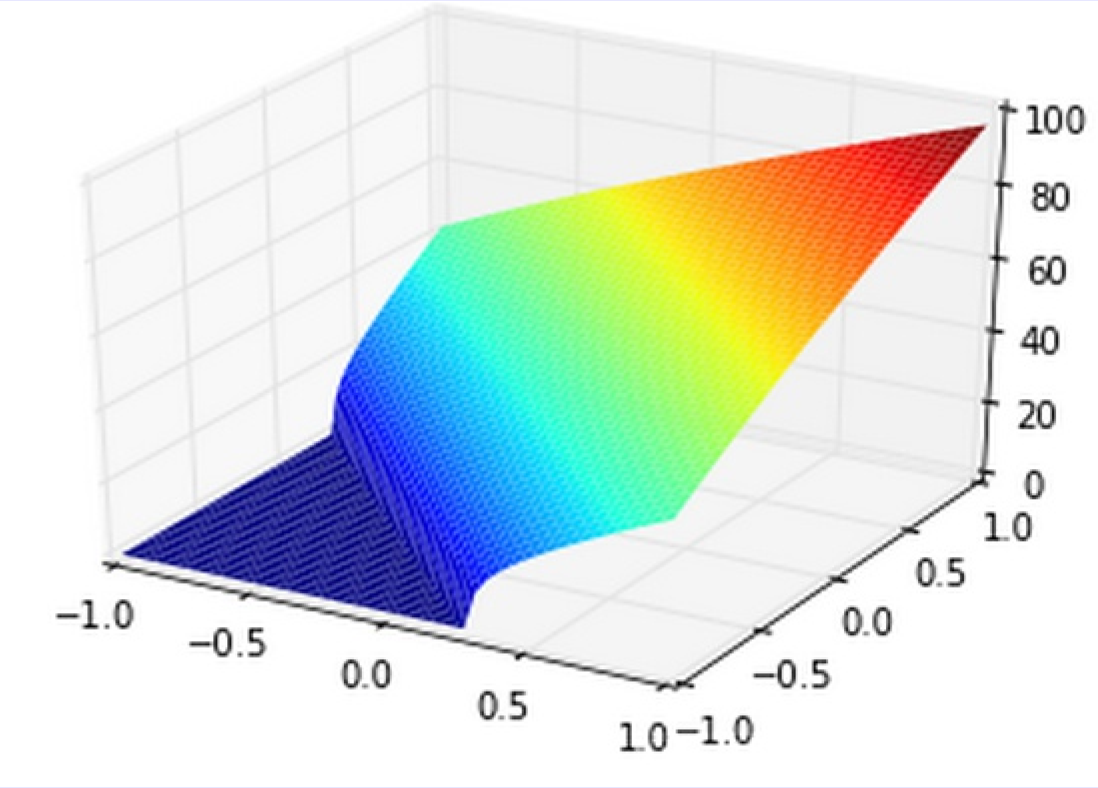
Tuning Curves

- Here is the general form we use for everything (it has both 'mappings' in it)
- $a_i = G_i[\alpha_i x \cdot e_i + J_i^{bias}]$
  - α is a gain term (constrained to always be positive)
  - J^{bias} is a constant bias term
  - e is the *encoder*, or the *preferred direction vector*
  - G is the neuron model
  - i indexes the neuron
- To simplify life, we always assume e is of unit length
  - Otherwise we could combine α and e
- In the 1D case, e is either +1 or -1
- In higher dimensions, what happens?

Tuning Curves

- 2-dimensional *x*

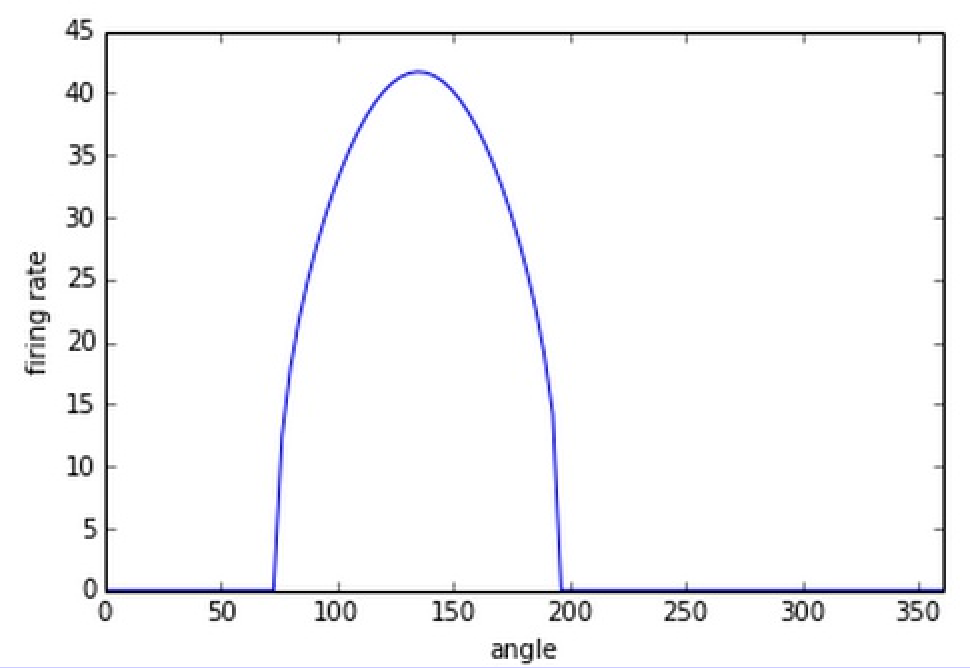
$$a_i = G_i\left[\alpha_i x \cdot e_i + J_i^{bias}\right]$$

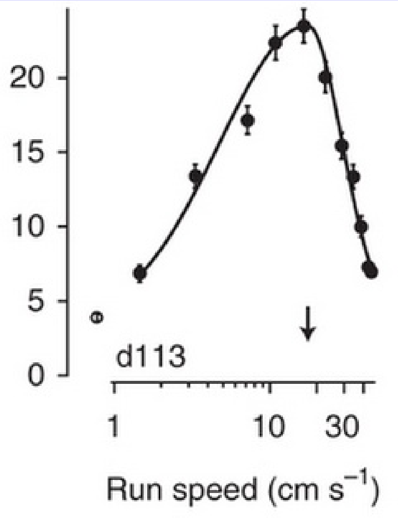


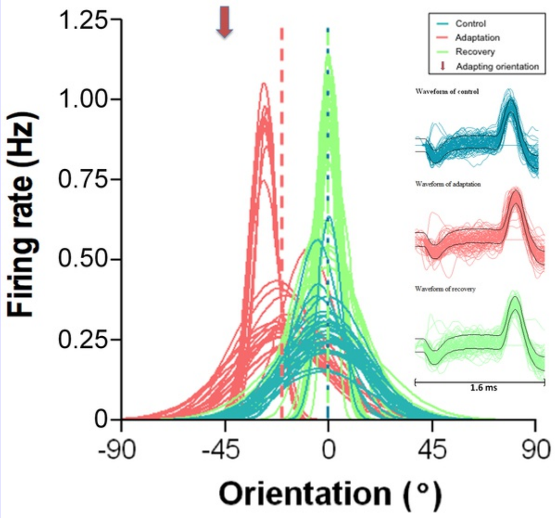
- But that's not how people normally plot it
- It might not make sense to sample *every possible* x
- Instead they might do some subset
  - For example, what if we just plot the points around the unit circle?

Tuning Curves

- Just along the unit circle

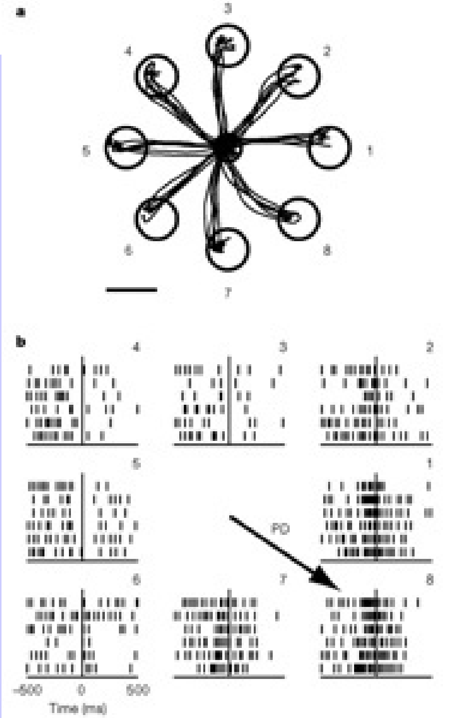


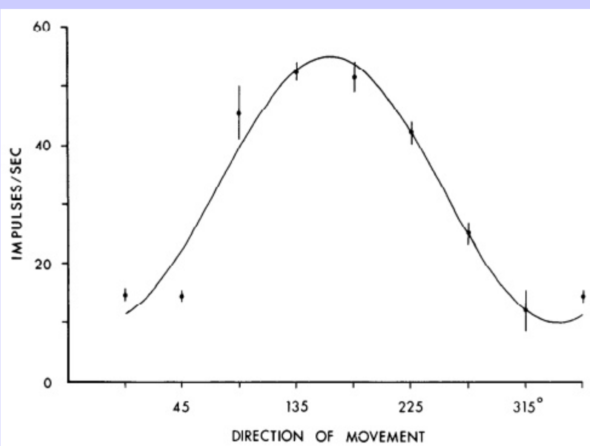




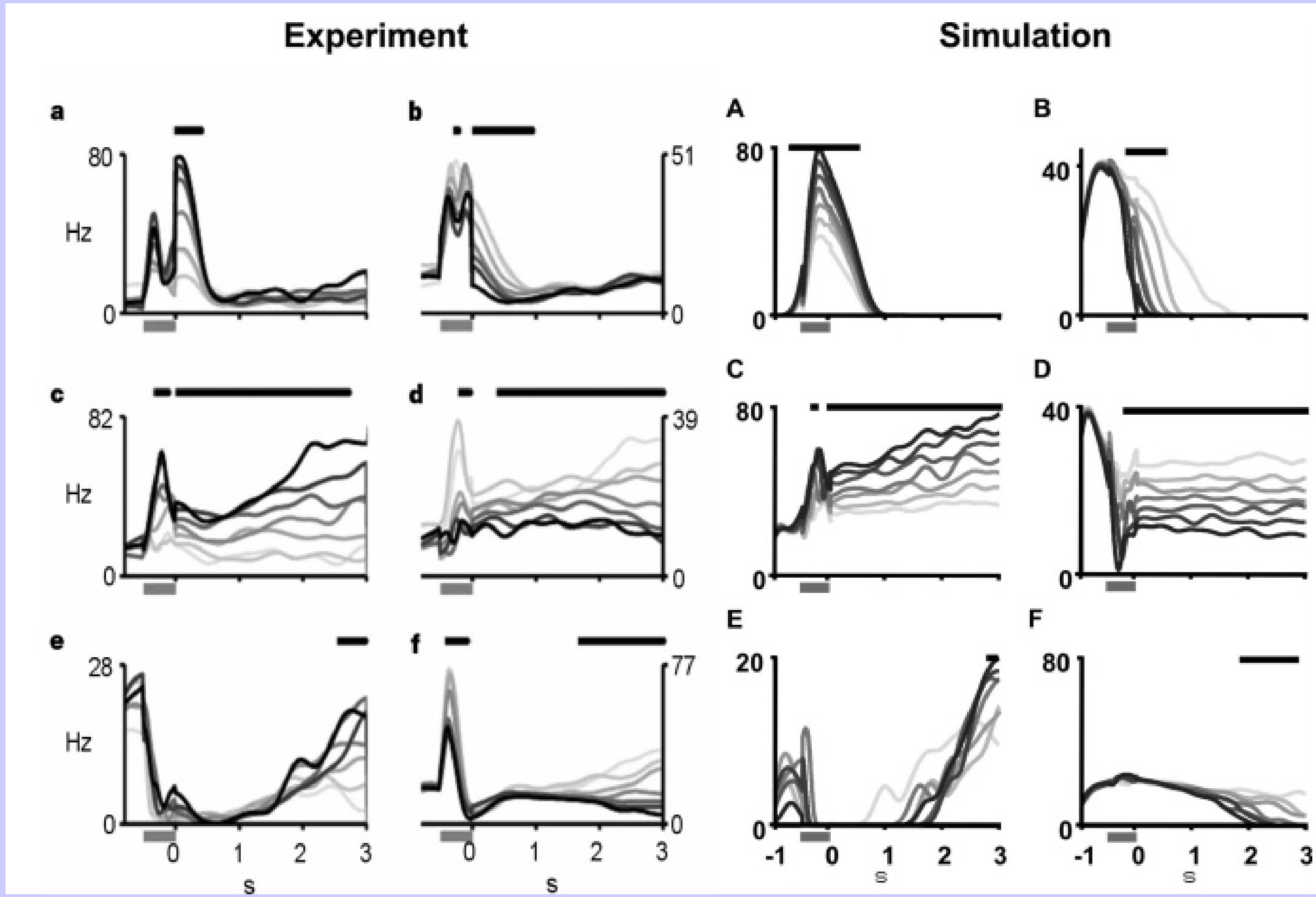
$$a_i = G_i[\alpha_i x \cdot e_i + J_i^{bias}]$$

Tuning Curves





(Georgopoulos et al., 1982)



(Singh & Eliasmith, 2005)

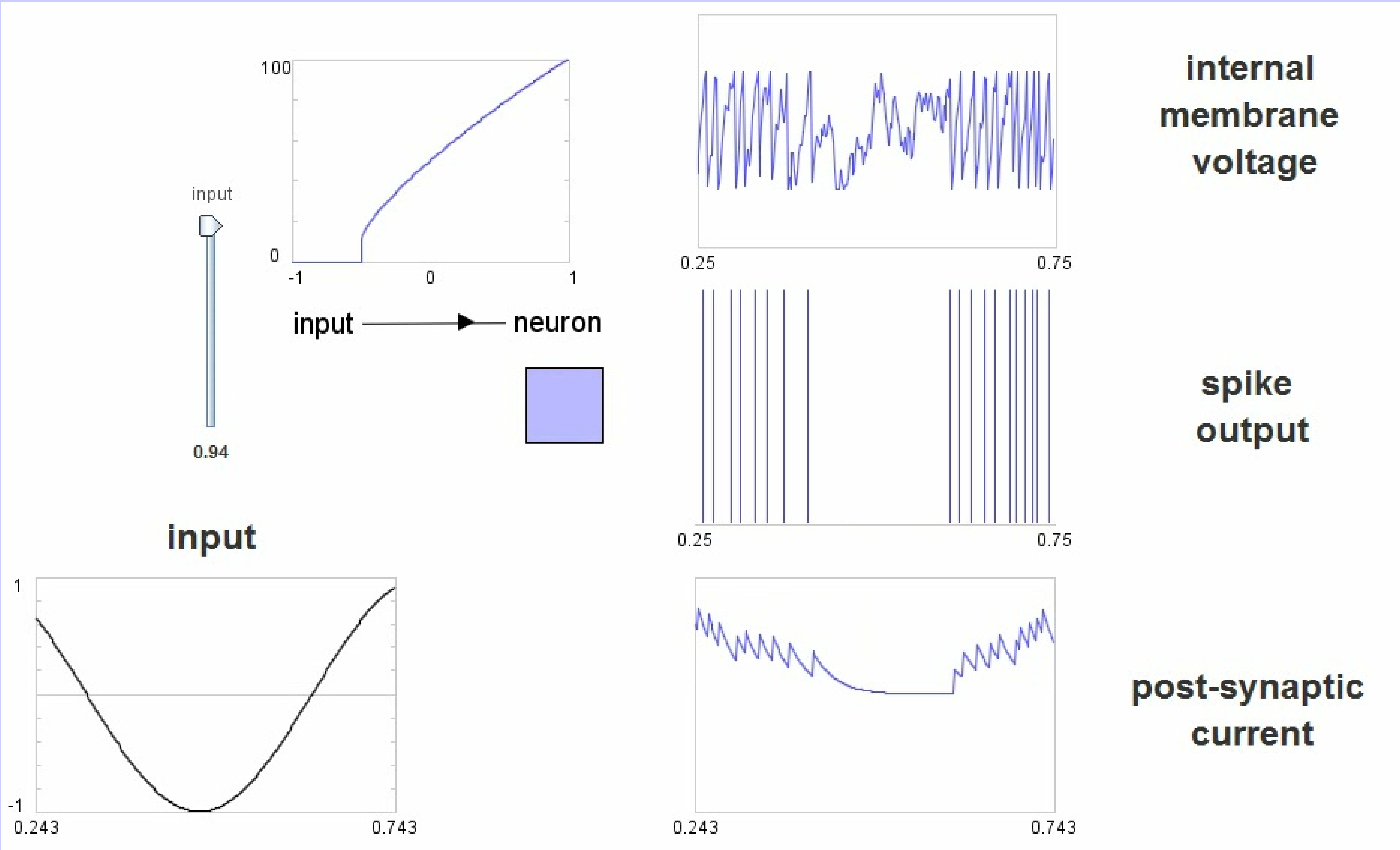
Tuning Curves

- General claim
  - For any neural data, we can chose a space *x* such that we can match the neural data using

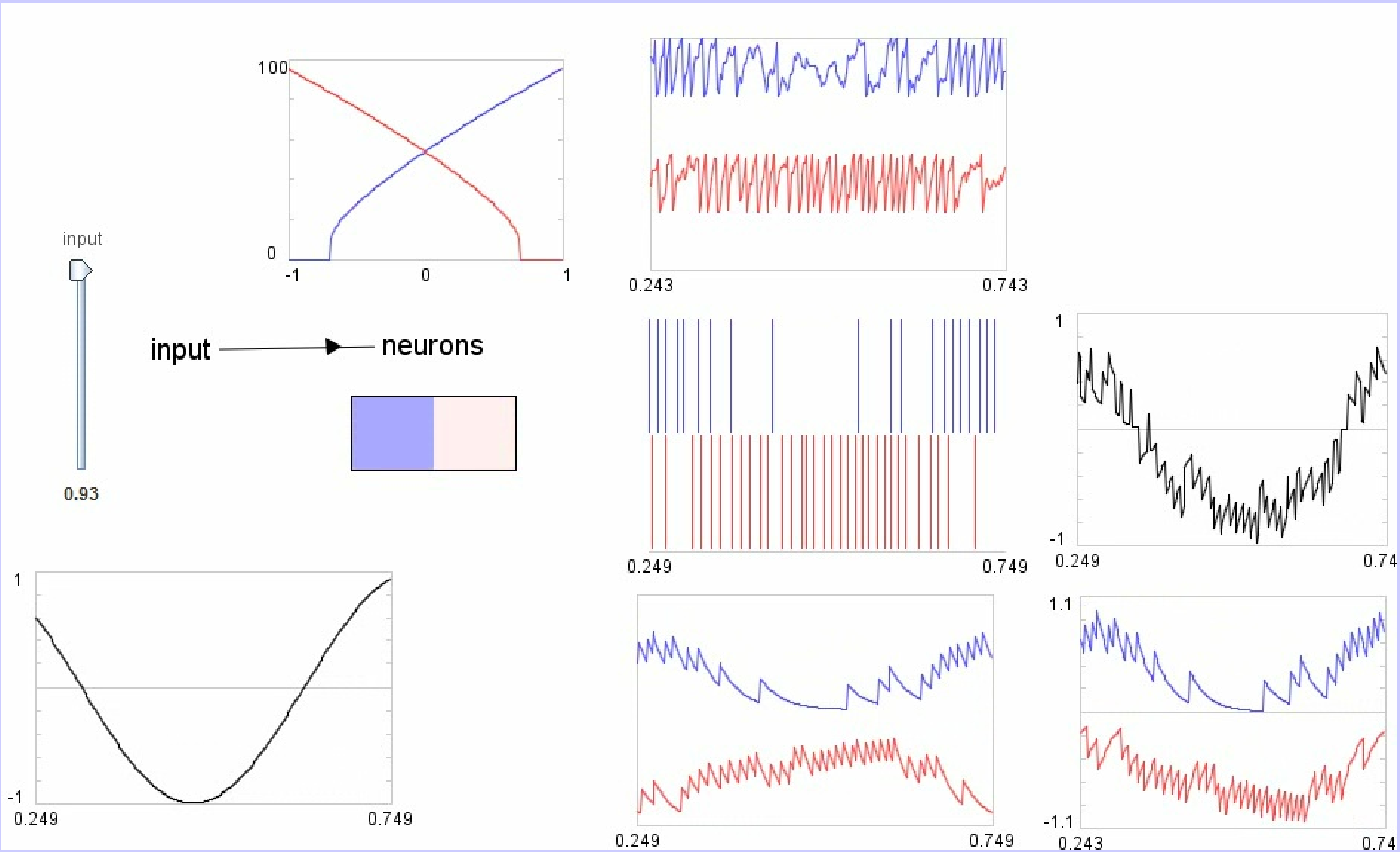
$$a_i = G_i[\alpha_i x \cdot e_i + J_i^{bias}]$$

  - Note: we don't need this assumption for NEF to work. We just need some map from x to a.
    - But this form seems to work well
    - Many neurons respond to multiple things
    - And gives us a really interesting shortcut soon
  - Note: what is *e*, physically?
- So what can we do with these neurons?

Single Neuron

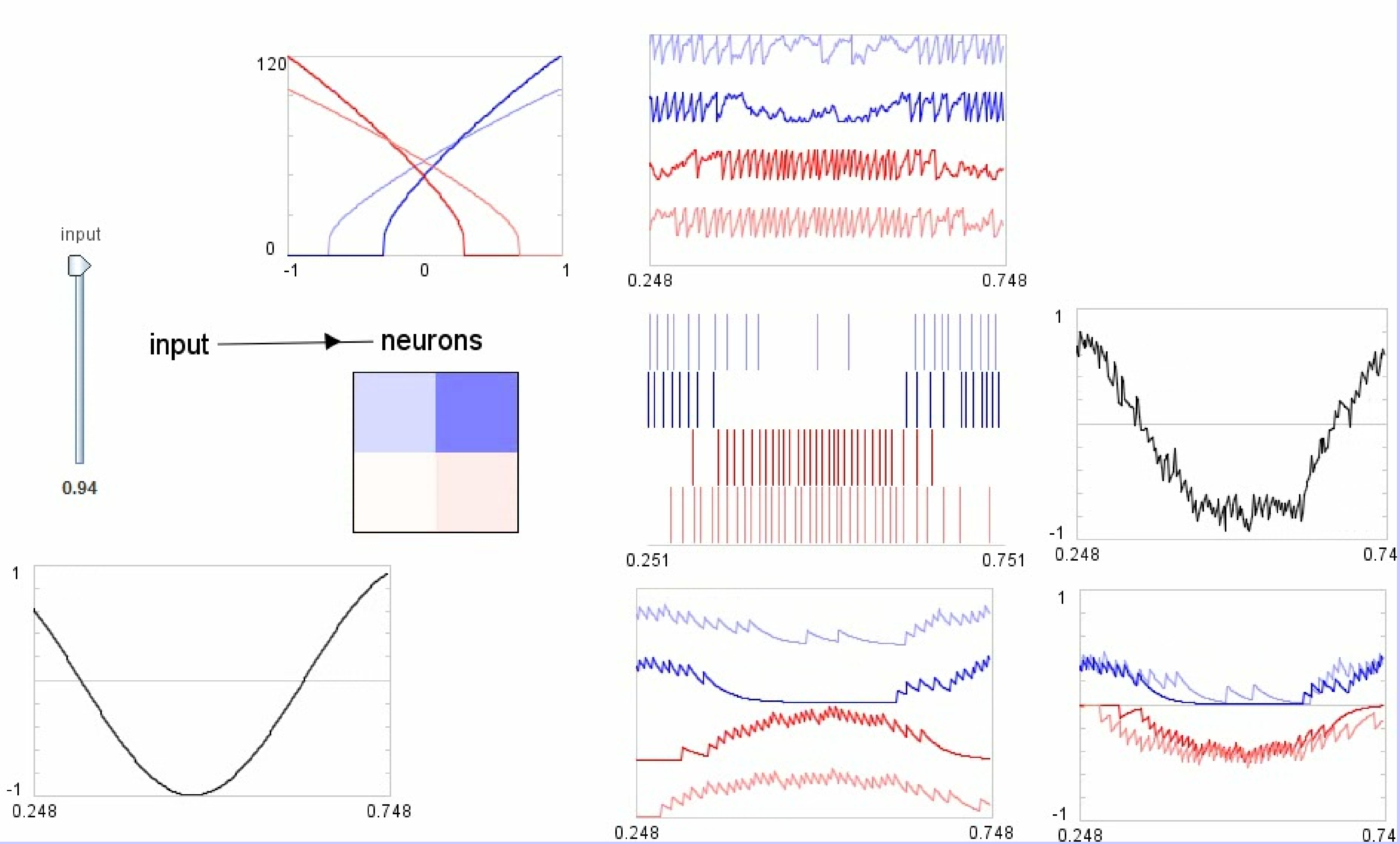


Two Neurons

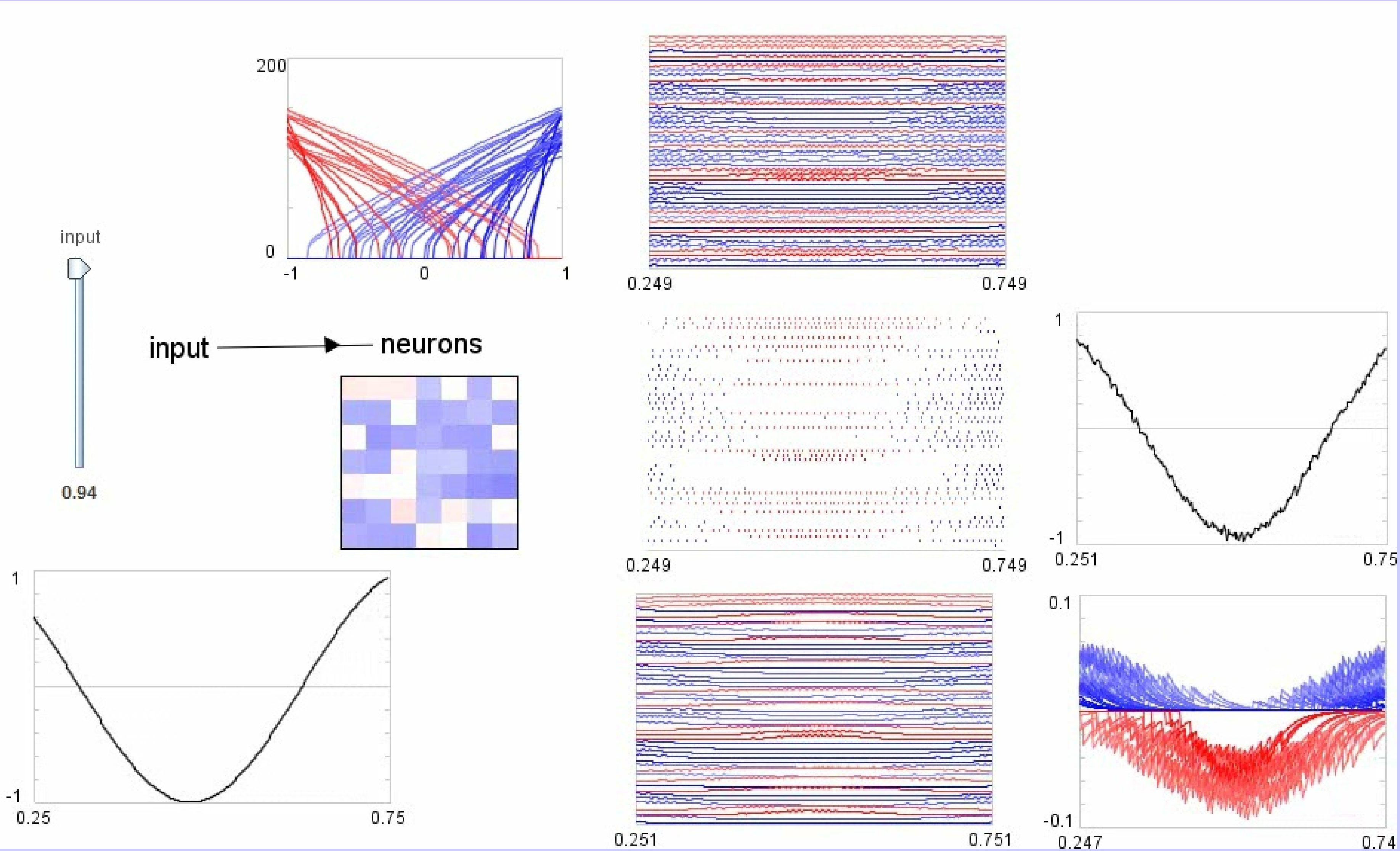


100
0
-1
0
1
0.243
0.743
input
0.93
input
neurons
0.249
0.749
1
-1
0.249
0.74
1
-1
0.249
0.749
0.249
0.749
1.1
-1.1
0.243
0.74

Four Neurons



Fifty Neurons



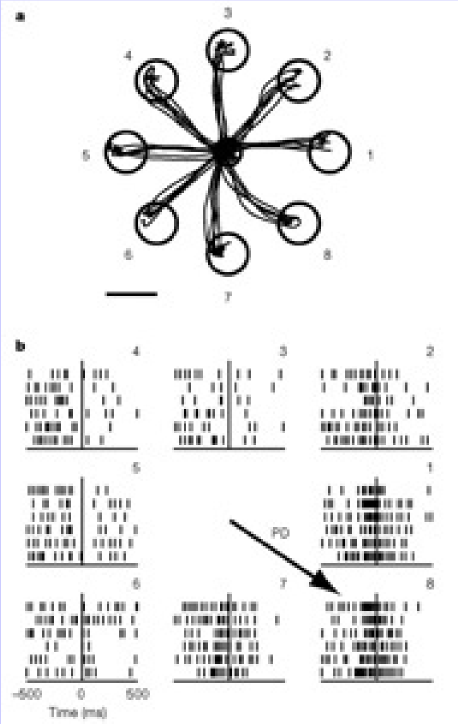
Neural computation

- Basic process
  - A group of neurons stores some value x
  - Each neuron has some preferred stimulus e_i
  - Current entering a neuron $J_i = x \cdot e_i$
  - Neurons fire based on their input $a_i = G_i[J_i]$
  - Decode output by weighted sum $\hat{x} = \sum_i a_i d_i$
  - Find decoders by minimizing error $E = (x - \sum a_i d_i)^2$
- Extensions
  - x can be a scalar or a vector
  - decoders for different functions $E = (f(x) - \sum a_i d_i)^2$

- [nengo example – scalar representation]

Multiple Dimensions

- Each neuron has a preferred direction (not just -1 or +1)
- Different weightings decode different values

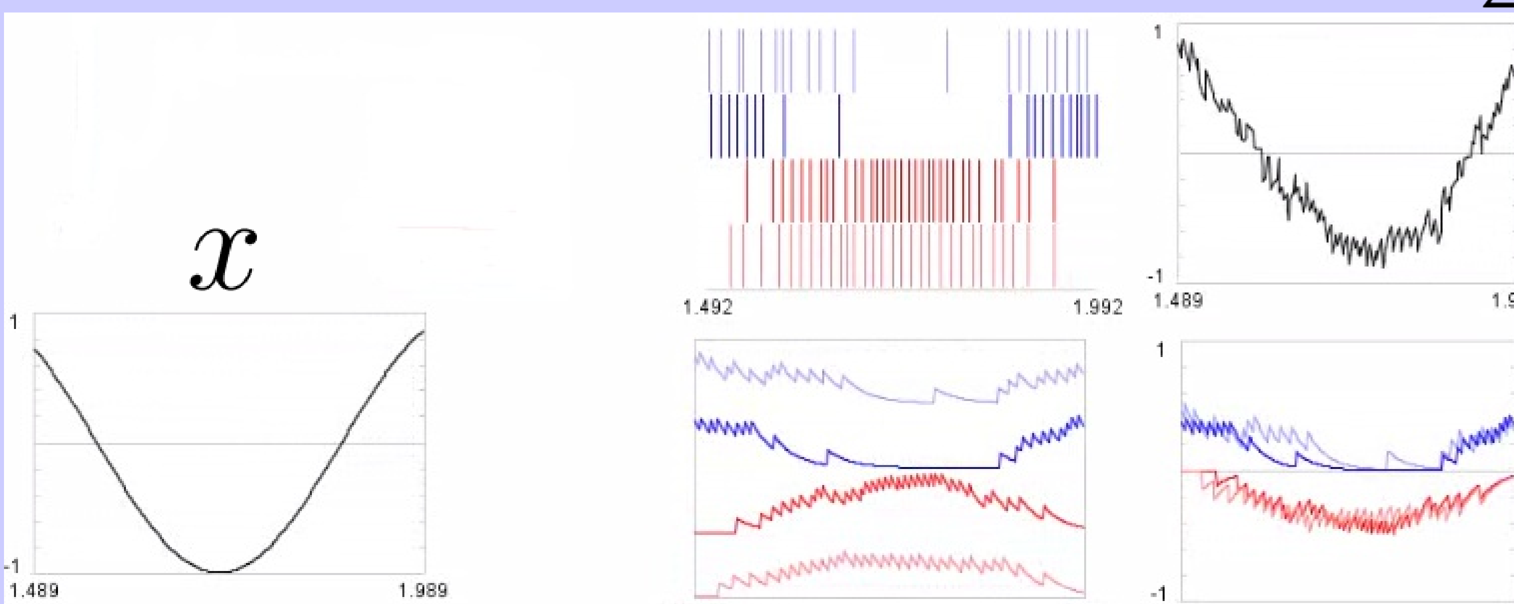


- [nengo example – 2d representation]

Decoders

- But where do we get d_i from?
  - $\hat{x} = \sum(a_i d_i)$
- Find the optimal d_i
  - How?

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



$$a_i d_i$$

Decoders

- But where do we get d_i from?
  - $\hat{x} = \sum (a_i d_i)$
- Find the optimal d_i
  - How?

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

- Take the derivative with respect to d_i

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

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

- At the minimum, $\frac{\partial E}{\partial d_i} = 0$

$\int_{-1}^{1} a_i x dx = \int_{-1}^{1} \sum_j (a_j d_j a_i) dx$

$\int_{-1}^{1} a_i x dx = \sum_j (\int_{-1}^{1} a_i a_j dx) d_j$

Decoders

- That's a system of N equations and N unknowns
- In fact, we can rewrite this in matrix form

$$\Upsilon = \Gamma d$$

where

$$\Upsilon_i = \frac{1}{2} \int_{-1}^{1} a_i x dx$$

$$\Gamma_{ij} = \frac{1}{2} \int_{-1}^{1} a_i a_j dx$$

- Do we have to do the integral over all x?
    - Approximate the integral by sampling over x
    - S is the number of x values to use (S for samples)

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

$$\Upsilon = \Gamma d$$

where

$$\Upsilon_i = \sum_x a_i x/S$$

$$\Gamma_{ij} = \sum_x a_i a_j/S$$

- Notice that if A is the matrix of activities (the firing rate for each neuron for each x value), then $\Gamma = A^T A/S$ and $\Upsilon = A^T x/S$

So given

$$\Upsilon = \Gamma d$$

then

$$d = \Gamma^{-1}\Upsilon$$

or, equivalently

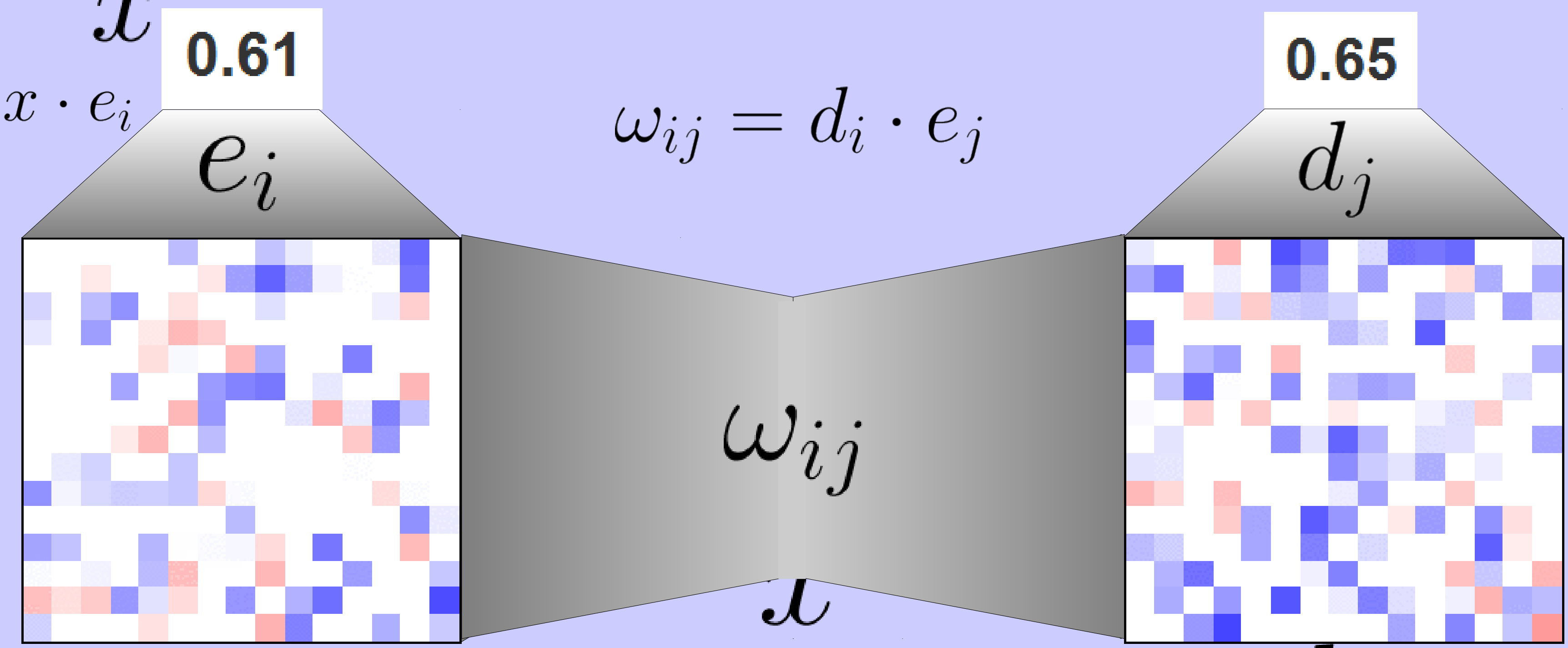
$$d_i = \sum_j \Gamma_{ij}^{-1}\Upsilon_j$$

Connecting Neurons

x

$J_i = x \cdot e_i$

$\omega_{ij} = d_i \cdot e_j$



a_i

$J_j = \sum_i \omega_{ij} a_i$

b_j

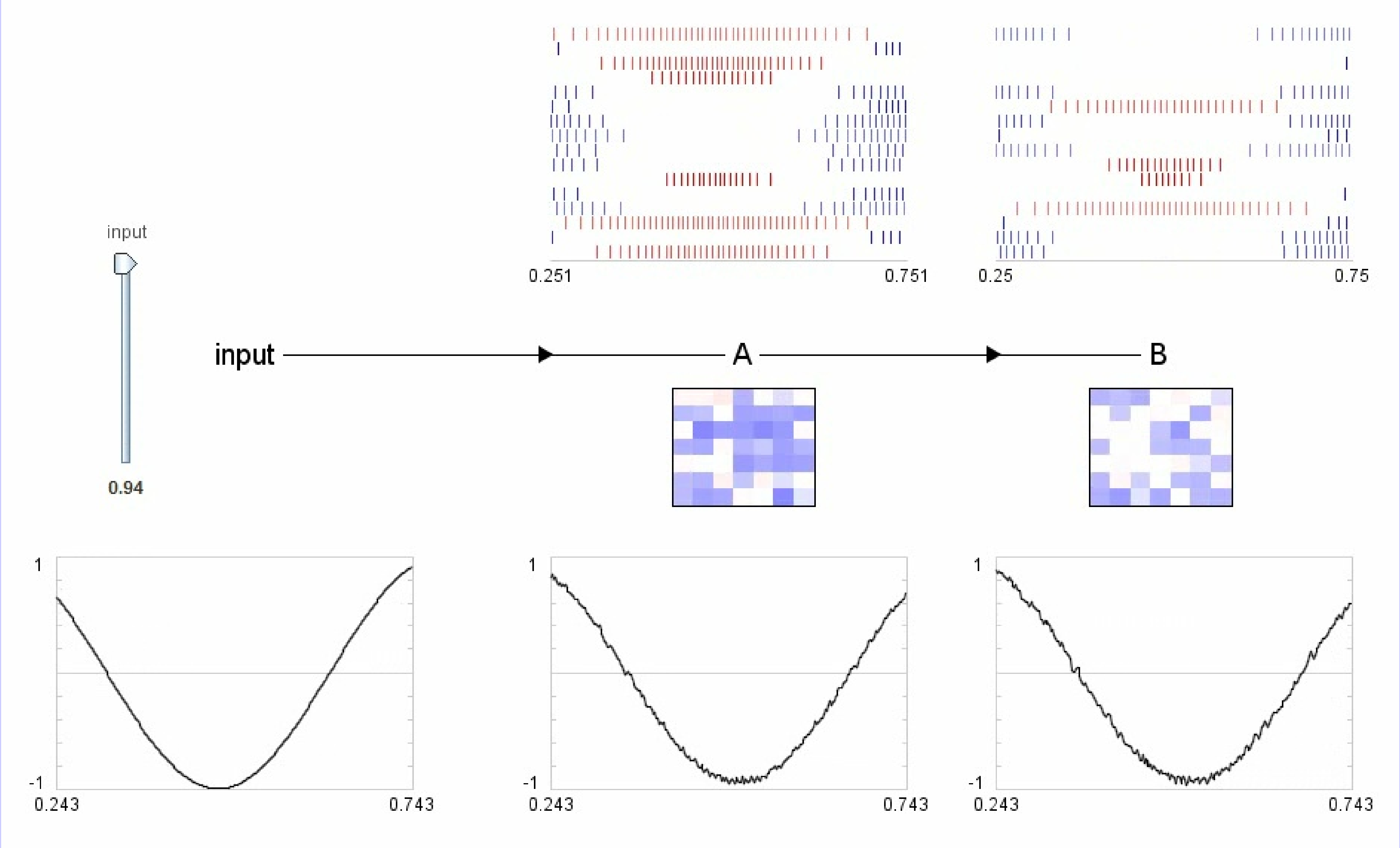
$a_i = G_i[J_i]$

$b_j = G_j[J_j]$

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

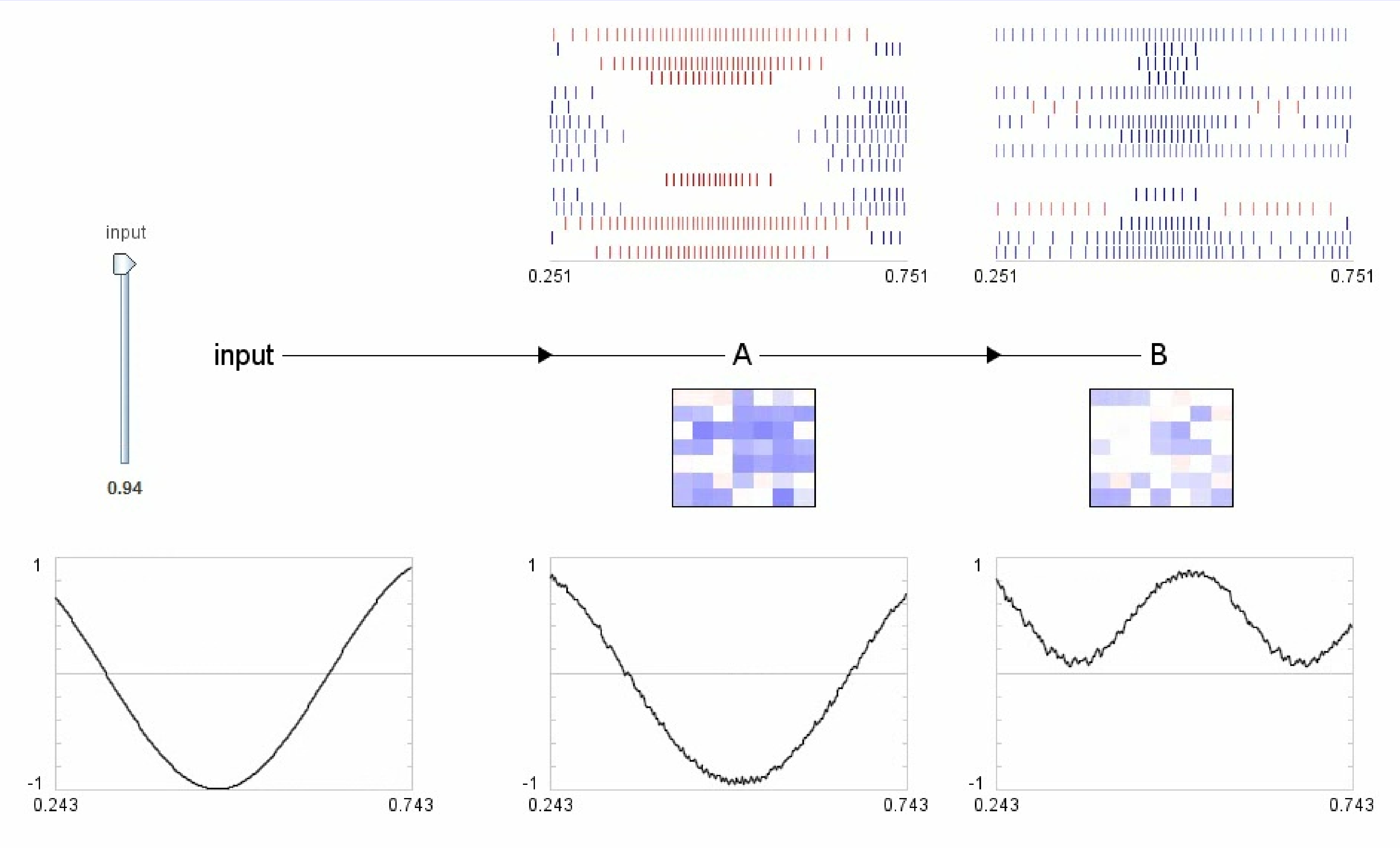
$J_j = \hat{x} \cdot e_j$

Communication Channel



- [nengo example - communication]

Computation



Computing $y=x^2$ requires the same amount of effort as $y=x$

Neural computation

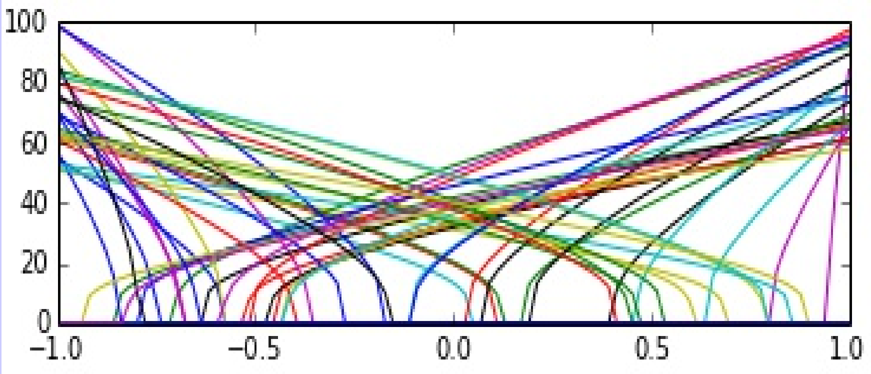
- [Nengo example: decoding functions]

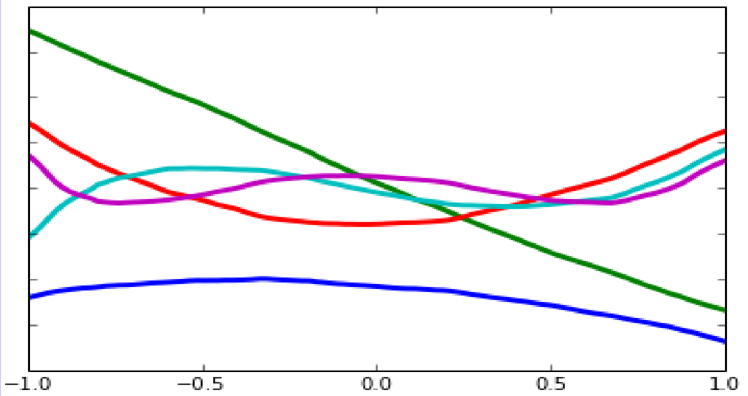
Computation

- Addition?
  - [nengo example]
- Combination?
  - [nengo example]
- Multiplication?
  - [nengo example]

Neural Computation

- With enough neurons, we can approximate any function to any degree of accuracy
  - $MSE \propto 1/N$
- What functions are neurons good at approximating?
  - Do SVD on the tuning curves

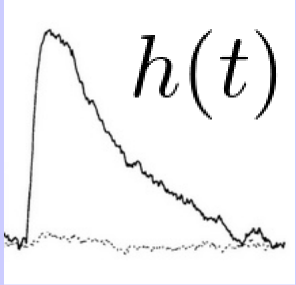


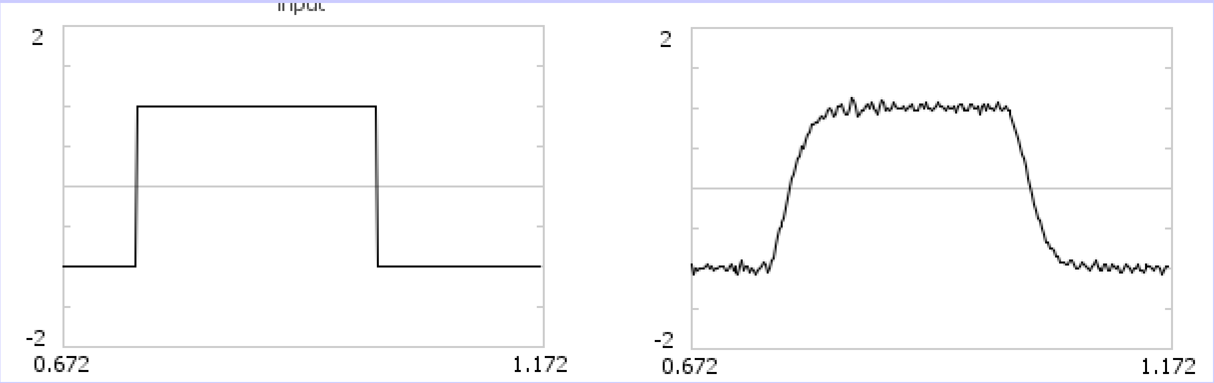


  - Low-degree polynomials (Legendre basis)

Computation

- Estimate any function $f(x)$
  - Accuracy increases as # neurons increases
  - Best at low-degree polynomials
- Not quite a perfect version of the function
  - Random noise due to neural activity
  - Smoothed due to post-synaptic current (varies from ~2ms to ~200ms)





$$f(x(t)) * h(t)$$

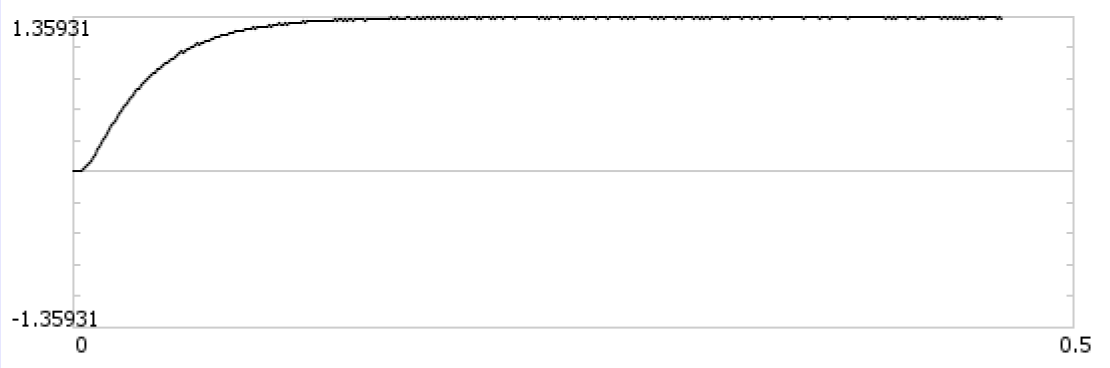
Biological Algorithms

- What do neural algorithms look like?
  - Each node (group of neurons) stores a vector
  - Each connection computes a function
    - and applies a filter
    - (set of functions and filter depends on neuron model)
- Different from standard connectionism
  - There, connections can only do linear weights
  - Some functions are easier than others
    - max(a,b) takes a very large number of neurons
    - sin(a+b)*cos(b - a) is pretty easy

Recurrent connections

- What happens if a group of neurons connects back to itself?
  - Depends on what function is being computed on the connection

$f(x) = x + 1$



$f(x) = -x$

$f(x) = x^2$

Nengo

- Open-source (free for non-commercial use)
  - http://nengo.ca
  - http://github.com/nengo/nengo
- Requirements
  - Python (2.7, 3.4, or 3.5)
  - NumPy
- Install
  - "pip install nengo"
  - "pip install nengo_gui"