# NONLINEAR TRANSFORMATIONS

Note: Projects...

# Nonlinearities

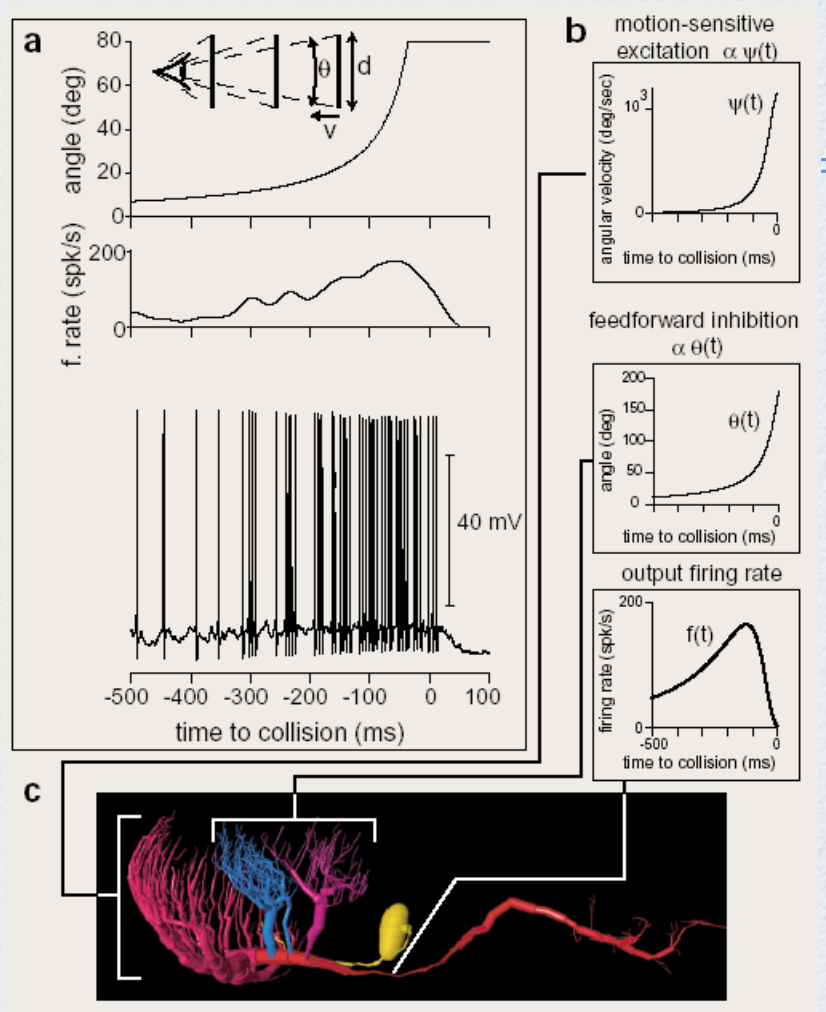
- Common in the nervous system (as we know)
- Generally ‘small’ nonlinearities (near linear)
- ‘Big’ nonlinearities are essential for useful information processing
- Such nonlinearities in single neurons (e.g. locust visual system) and in networks (e.g. gain fields)
- Single cell nonlinearities are not yet indisputable (and their form is not well understood)

# Locust visual system

Single cell multiplication
Firing rate is given by

$$f(t) = \theta'(t-\delta) * e^{-\alpha * \theta(\tau - \delta)}$$

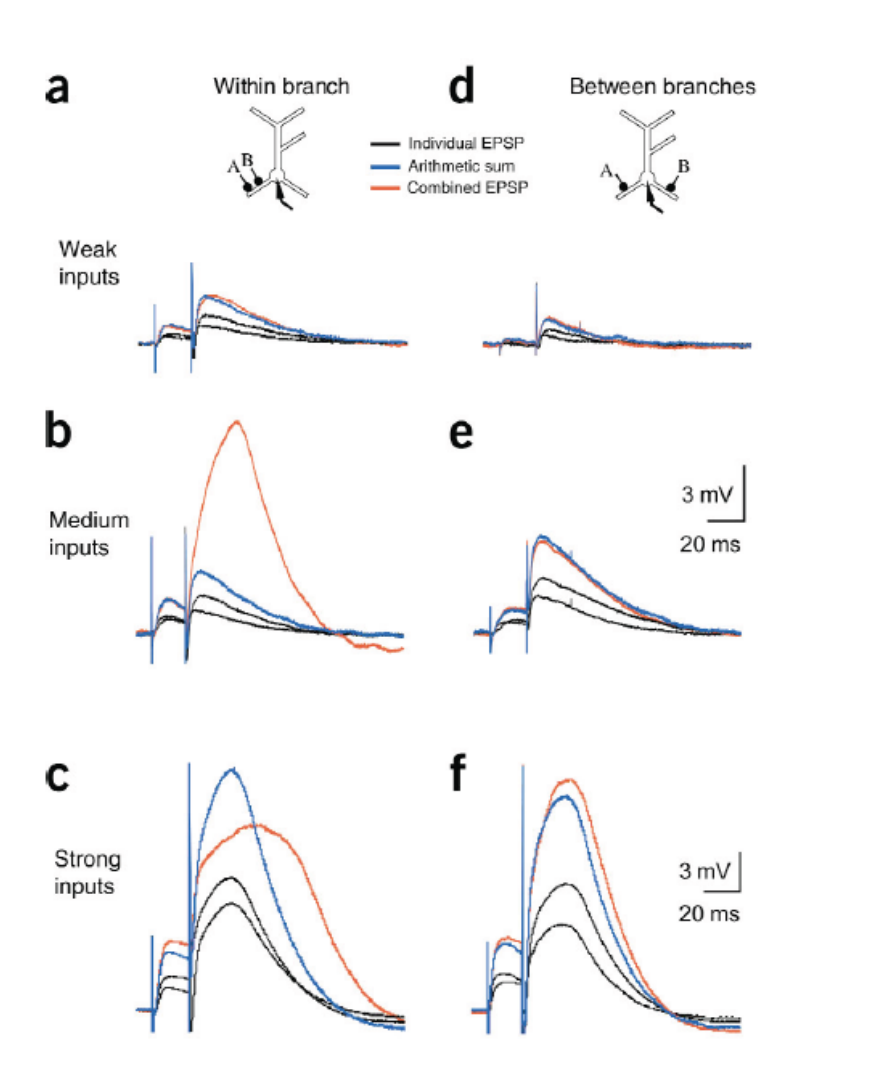
Mechanism unknown



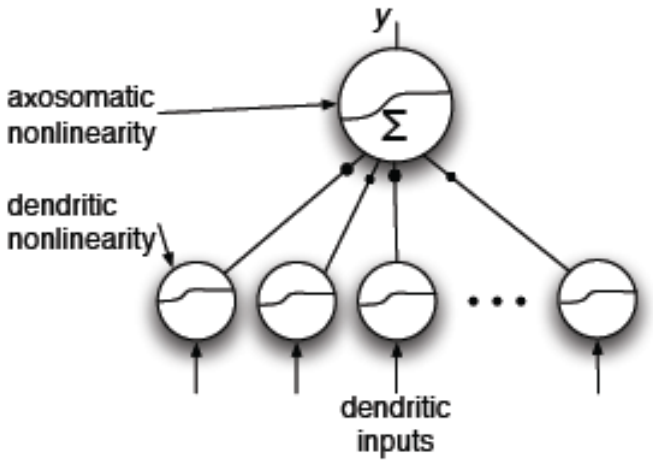

=θ'

From Koch and Segev, 2000, Nat. Neuro

# Single cell nonlin. in cortex




Bartlett Mel and colleagues have found evidence for nonlinearities in single neurons in cortex (pyramidal cells).



# Recipe for linear trans.

- 1. Define the repn (enc/dec) for all variables involved in the operation.
- 2. Write the transformation in terms of these variables.
- 3. Write the transformation using the decoding expressions for all variables except the output variable.
- 4. Substitute this expression into the encoding expression of the output variable.

# Product of variables

- Follow the recipe (volunteer)
- Representation:

$$\begin{aligned} a_i(x) &= G_i\left[\alpha_i \left\langle \tilde{\phi}_i x \right\rangle + J_i^{bias}\right] \\ \hat{x} &= \sum_i a_i(x)\phi_i^x \end{aligned}$$

- Transformation

$$\begin{aligned} c_k(x \cdot y) &= G_k\left[\alpha_k \tilde{\phi}_k (x \cdot y) + J_k^{bias}\right] \\ &= G_k\left[\alpha_k \tilde{\phi}_k \left(\sum_i a_i(x)\phi_i^x \cdot \sum_j b_j(y)\phi_j^y\right) + J_k^{bias}\right] \\ &= G_k\left[\sum_{ij} \omega_{kij} a_i(x) b_j(y) + J_k^{bias}\right] \end{aligned}$$

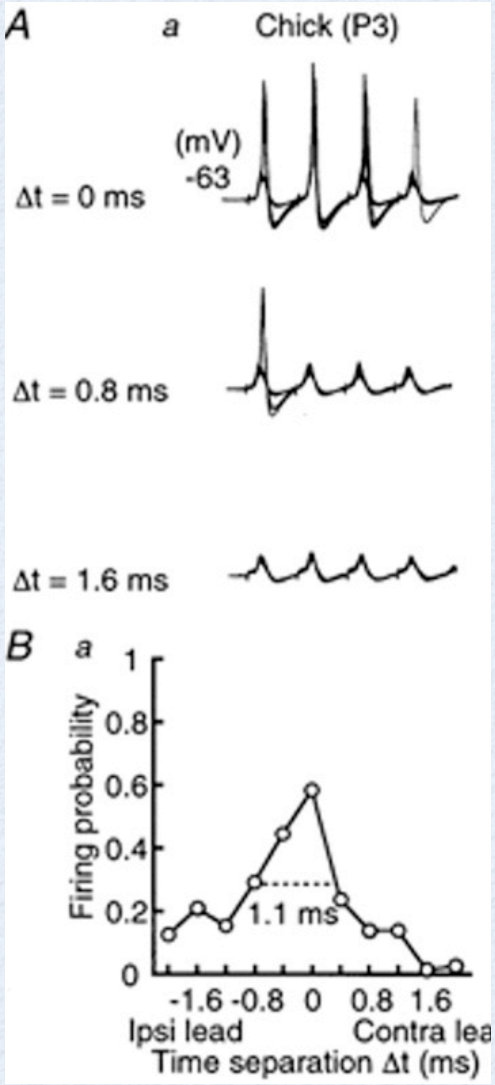
$$\omega_{kij} = \alpha_k \tilde{\phi}_k \phi_i^x \phi_j^y$$

# Single cell nonlinearities

- Need individual neurons computing nonlinearities
- One proposed mechanism for implementing this kind of multiplication in neurons is
  - ‘coincidence detection’

# Coincidence detection

Input to two dendrites applied at different delays.

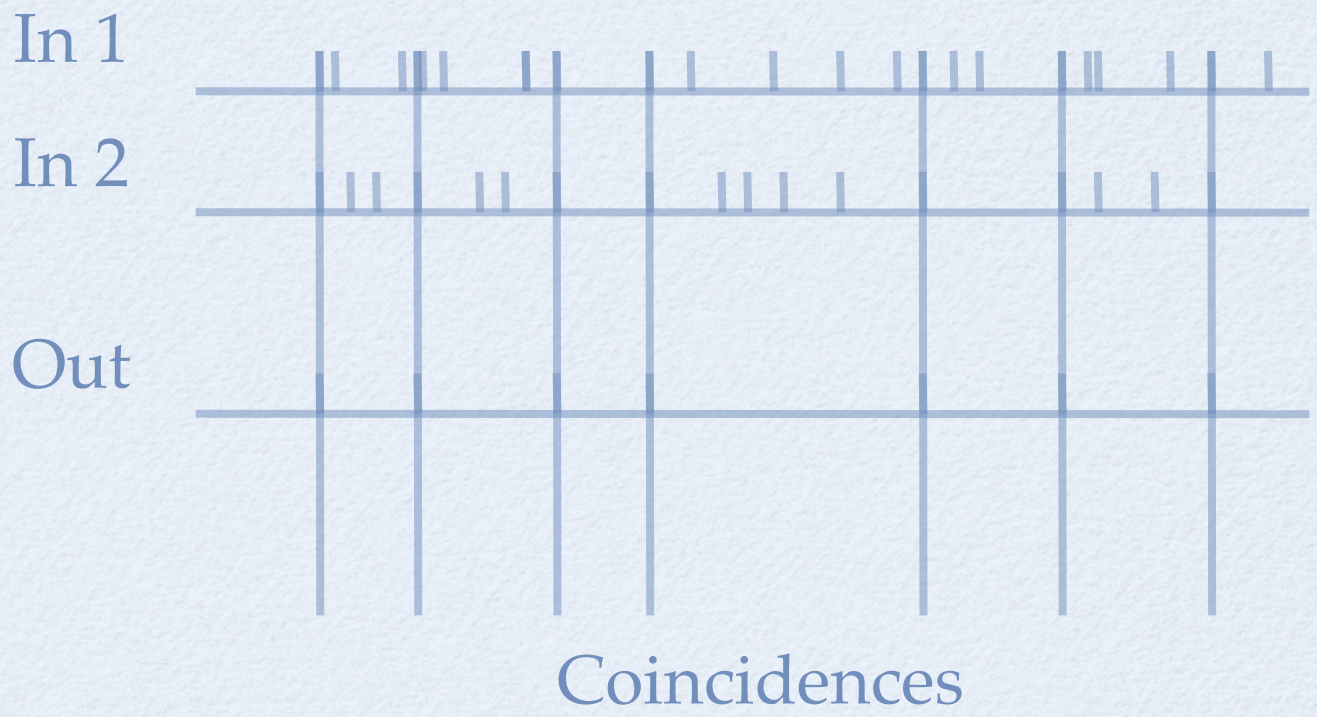



A These graphs are 10 superimposed traces.

Hiroshi Kuba et al Development of membrane conductance improves coincidence detection in the nucleus laminaris of the chicken, J Physiol April 15, 2002 vol. 540 no. 2 529-542

B Firing probability calculated from 40 stimuli.

# Co-incidence detection



# Coincidence detection

- Let's look at the PSCs

$$a_i(x)b_j(y) = \sum_{n,m} h_i(t - t_{in})h_j(t - t_{jm})$$

- For convenience let $h(t)$ be small Gaussians

$$\begin{aligned} a_i(x)b_j(y) &= \sum_{n,m} e^{-(t-t_{in})^2/2\Delta^2} e^{-(t-t_{jm})^2/2\Delta^2} \\ &= \sum_{n,m} e^{-[(t-t_{in})^2+(t-t_{jm})^2]/2\Delta^2} \end{aligned}$$

# Coincidence detection

- After a little manipulation, we have

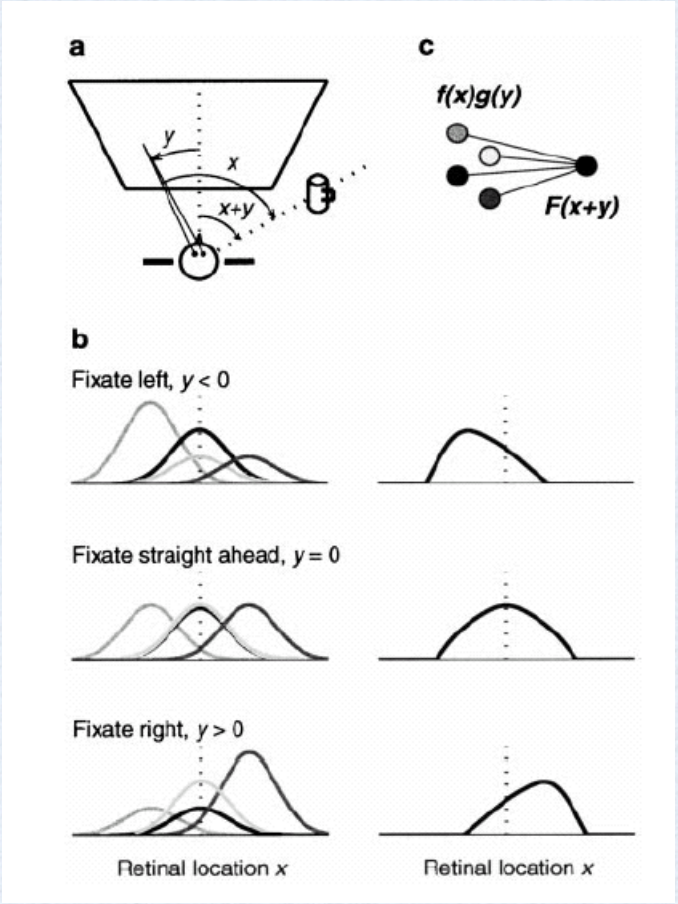
$$a_i(x)b_j(y) = \sum_{n,m} e^{-\left[2\left(t-\frac{t_{in}+t_{jm}}{2}\right)^2+\frac{1}{2}(t_{in}-t_{jm})^2\right]/2\Delta^2}$$

$$= \sum_{n,m} e^{-2\left(t-\frac{t_{in}+t_{jm}}{2}\right)^2/2\Delta^2} e^{-\frac{1}{2}(t_{in}-t_{jm})^2/2\Delta^2}$$

- So, multiplying PSCs is just like CD
- Drawback: every spike from each neuron must be compared to every other spike from each neuron... massively 'over connected' structure.

# Networks: Gain fields

- The rate of the (idealized) neurons in b) are R=f(x)g(y)
- x+y is needed to reach target
- Adding these multiplicatively modulated 'gain fields' gives an estimate




- From Salinas and Their, 2000, Neuron

# How to perform multiplication

- First, notice that we can compute nonlinear functions of encoded variables by finding "transformational decoders"

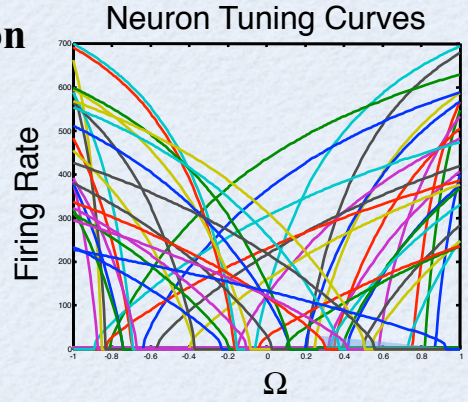
$$\begin{aligned} E &= \frac{1}{2}\langle[f(x) - \hat{f}(x)]^2\rangle \\ &= \frac{1}{2}\langle[f(x) - \sum_i a_i(x)\phi_i^{f(x)}]^2\rangle \end{aligned}$$

- As before: $\phi^{f(x)} = \Gamma^{-1}\Upsilon$
- But: $\Upsilon_i = \int a_i(x) f(x) dx$

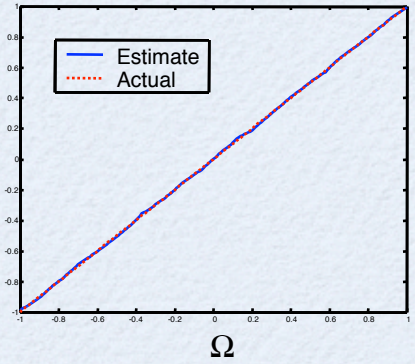
# Transformational decoders

**Representation**

Encoding

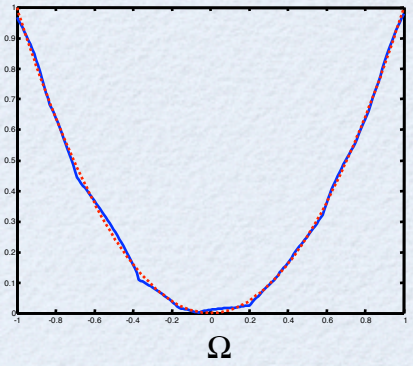



Decoding (Linearity)

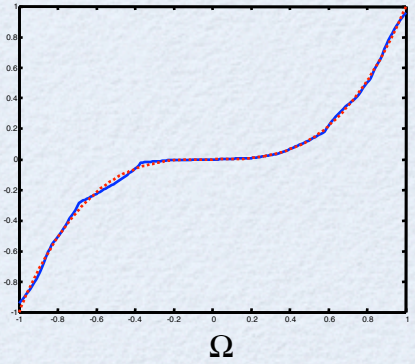



**Computation**

Quadratic

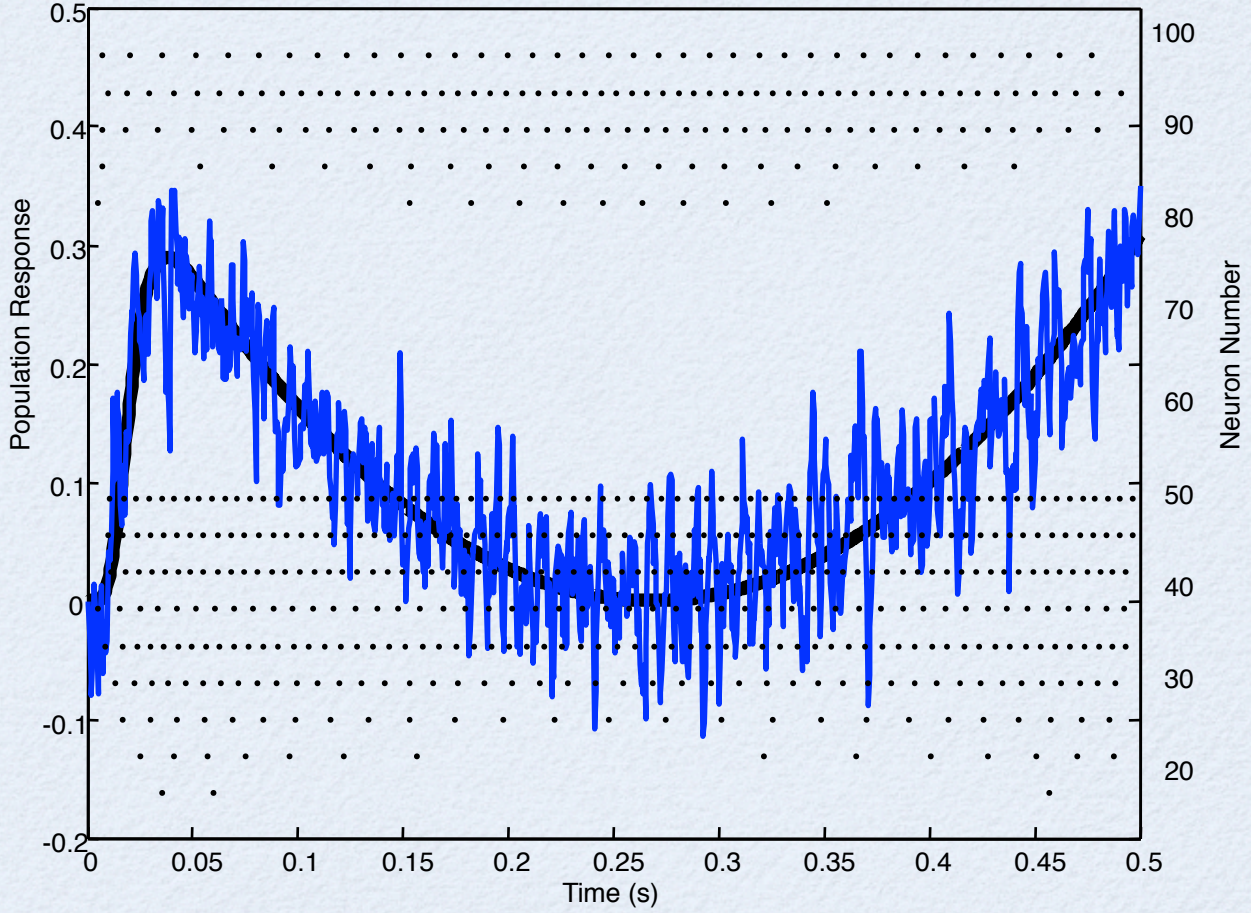



Cubic



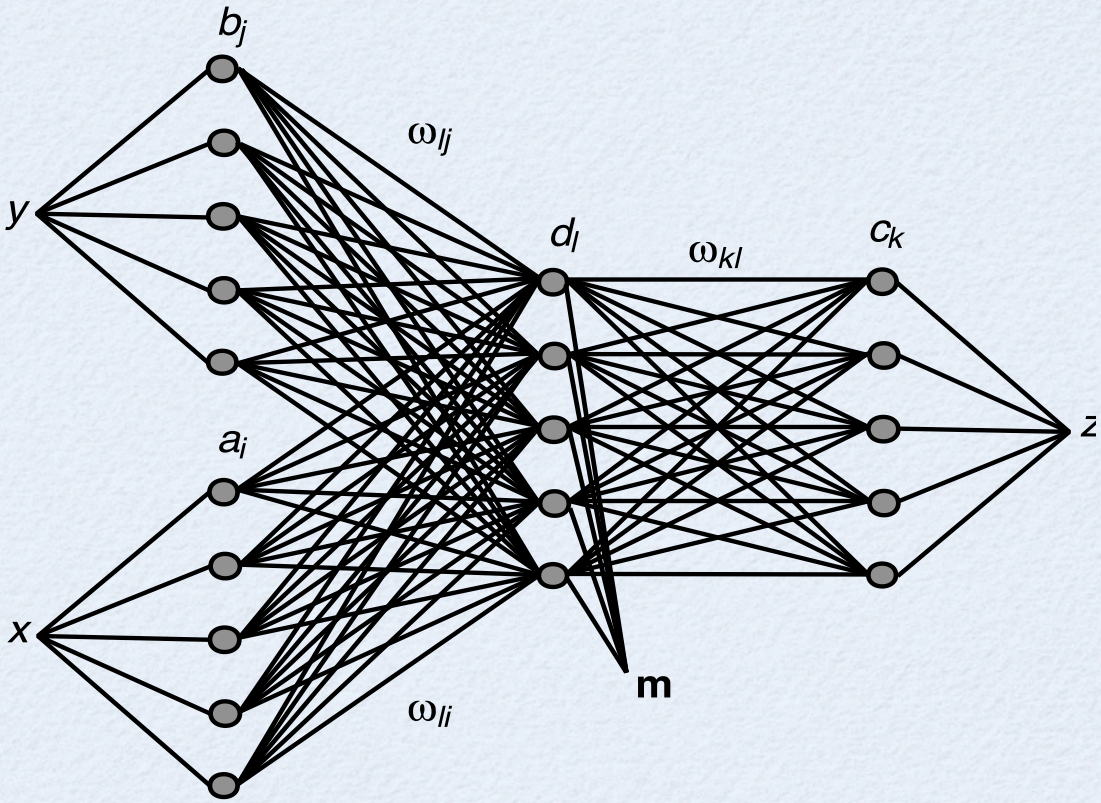
# Spiking network

- Can be implemented in a noisy spiking network (ramp input)



# Nonlinearities via networks

- Form an intermediate repn in a `middle layer' of neurons with dimensionality $D_m=D_x+D_y$



# Nonlinearities via networks

- Then find an optimal linear decoder to approximate the product

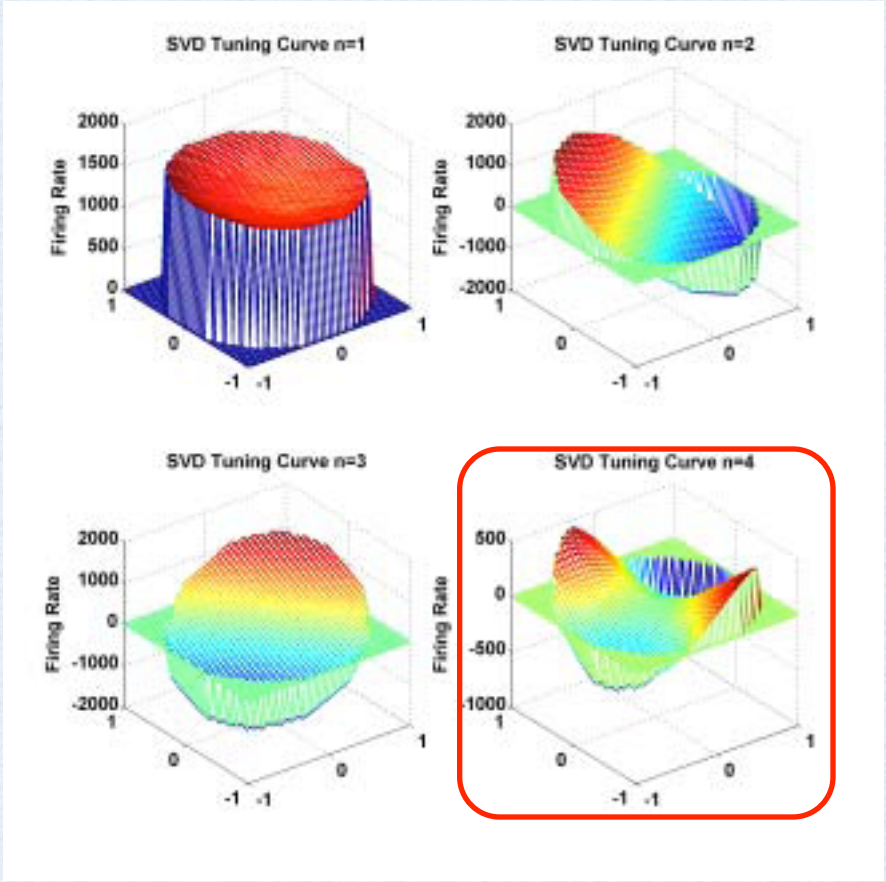
$$\hat{f}(\mathbf{m}) = \sum_l d_l(\mathbf{m})\phi_l^f$$

- By solving

$$E_f = \left\langle \left[ f(\mathbf{m}) - \hat{f}(\mathbf{m}) \right]^2 \right\rangle_{\mathbf{m}}$$

# Vector function decoders

- When we compute nonlinear functions of vectors, it is analogous to computing nonlinear functions in scalar spaces (though along any direction).



# Network nonlinearities

- First find the weights to put the inputs into this *m-D* space

$$
\begin{aligned}
d_l(\mathbf{m} = [x\, y]) &= G_l\left[\alpha_l \left\langle \tilde{\boldsymbol{\phi}}_l \mathbf{m} \right\rangle + J_l^b\right] \\
&= G_l\left[\alpha_l \left(\tilde{\phi}_l^{m_1} \hat{x} + \tilde{\phi}_l^{m_2} \hat{y}\right) + J_l^b\right] \\
&= G_l\left[\sum_i \omega_{li}^{m_1} a_i(x) + \sum_j \omega_{lj}^{m_2} b_j(y) + J_l^b\right],
\end{aligned}
$$

$$
\omega_{li}^{m_1} = \alpha_l \phi_i^x \tilde{\phi}_l^{m_1} \qquad \omega_{lj}^{m_2} = \alpha_l \phi_j^y \tilde{\phi}_l^{m_2}
$$

# Network nonlinearities

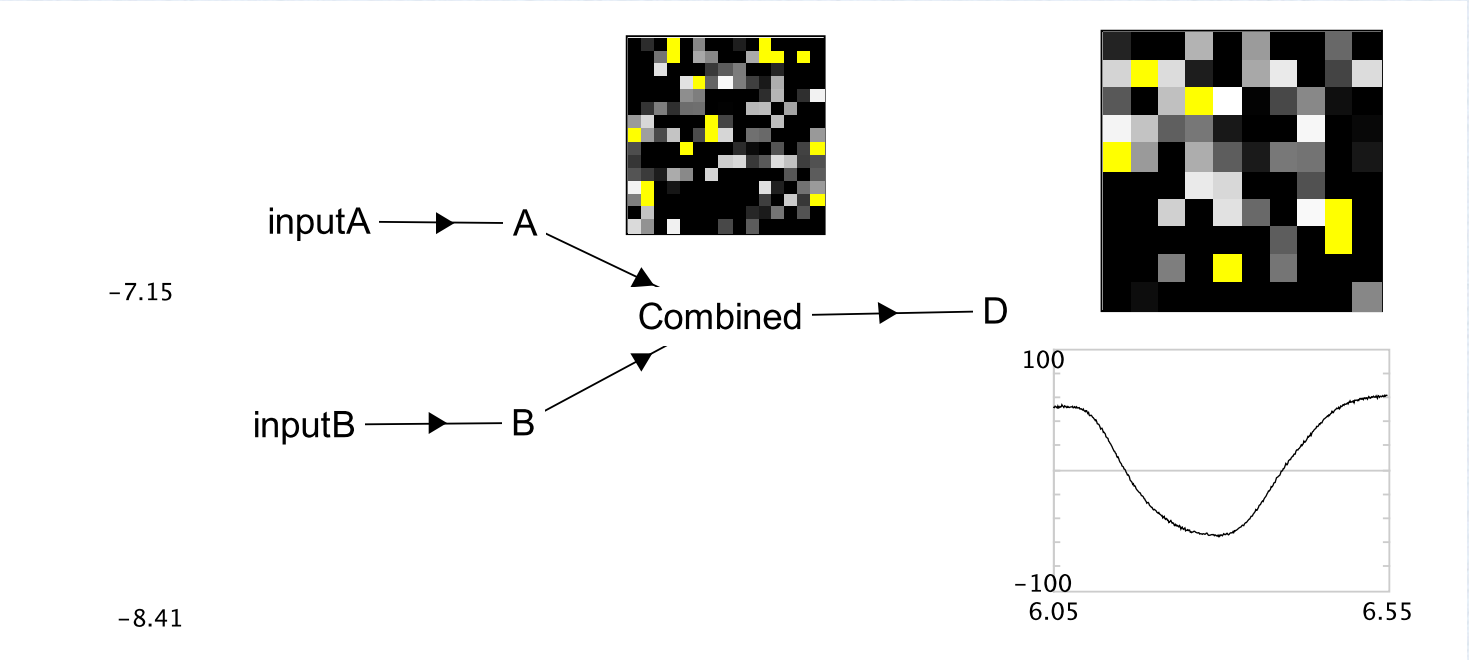
- Then use the transformation decoders we found earlier to extract the product

$$
\begin{aligned}
c_k(f(\mathbf{m})) &= G_k\left[\alpha_k\left(\tilde{\phi}_k f(\mathbf{m})\right) + J_k^b\right] \\
&= G_k\left[\alpha_k\left(\tilde{\phi}_k \sum_l d_l(\mathbf{m})\phi_l^f\right) + J_k^b\right] \\
&= G_k\left[\sum_l \omega_{kl} d_l(\mathbf{m}) + J_k^b\right]
\end{aligned}
$$

$$
\omega_{kl} = \alpha_k \tilde{\phi}_k \phi_l^f
$$

# Nengo implementation

- Run demo



# Comments

- Essentially, we have derived the `hidden layer' typical in ANNs. ANNs with such a layer can compute any function of the input
- Also suggests a way of including nonlinearities in dendrites: embed the **m** layer into the dendrites.
- Makes nonlinearities internal to the cell, and we need far fewer cells to do multiplication -- not too straightforward, however

# Negative weights

- Largely skipping this section
- Dale's principle: neurons are inhibitory or excitatory
- This provides a mapping from any mixed weight network to a 'biologically plausible' one while preserving the function
- Works for recurrent nets, and arbitrary dimensions (Nengo button)

# FUNCTION REPRESENTATION

# Representations we've seen

| *Neural System* | *Dimension* | *Encoder* ($a_i(x)$) | *Decoder* ($\hat{x}$) |
|---|---|---|---|
| Neural Integrator | 1-D scalar | $G_i\left[\alpha_i x + J_i^{bias}\right]$ | $\sum_i a_i(x)\phi_i$ |
| Motor Cortex | 2-D vector | $G_i\left[\alpha_i \left\langle \tilde{\boldsymbol{\phi}}_i \mathbf{x} \right\rangle_n + J_i^{bias}\right]$ | $\sum_i a_i(\mathbf{x})\boldsymbol{\phi}_i$ |
| Semicircular Canals | 2-D vector | $G_{im}\left[\alpha_{im} \left\langle \mathbf{x}\tilde{\boldsymbol{\phi}}_{im} \right\rangle_n + J_{im}^{bias}\right]$ | $\sum_{i,m} a_{im}(\mathbf{x})\boldsymbol{\phi}_{im}$ |

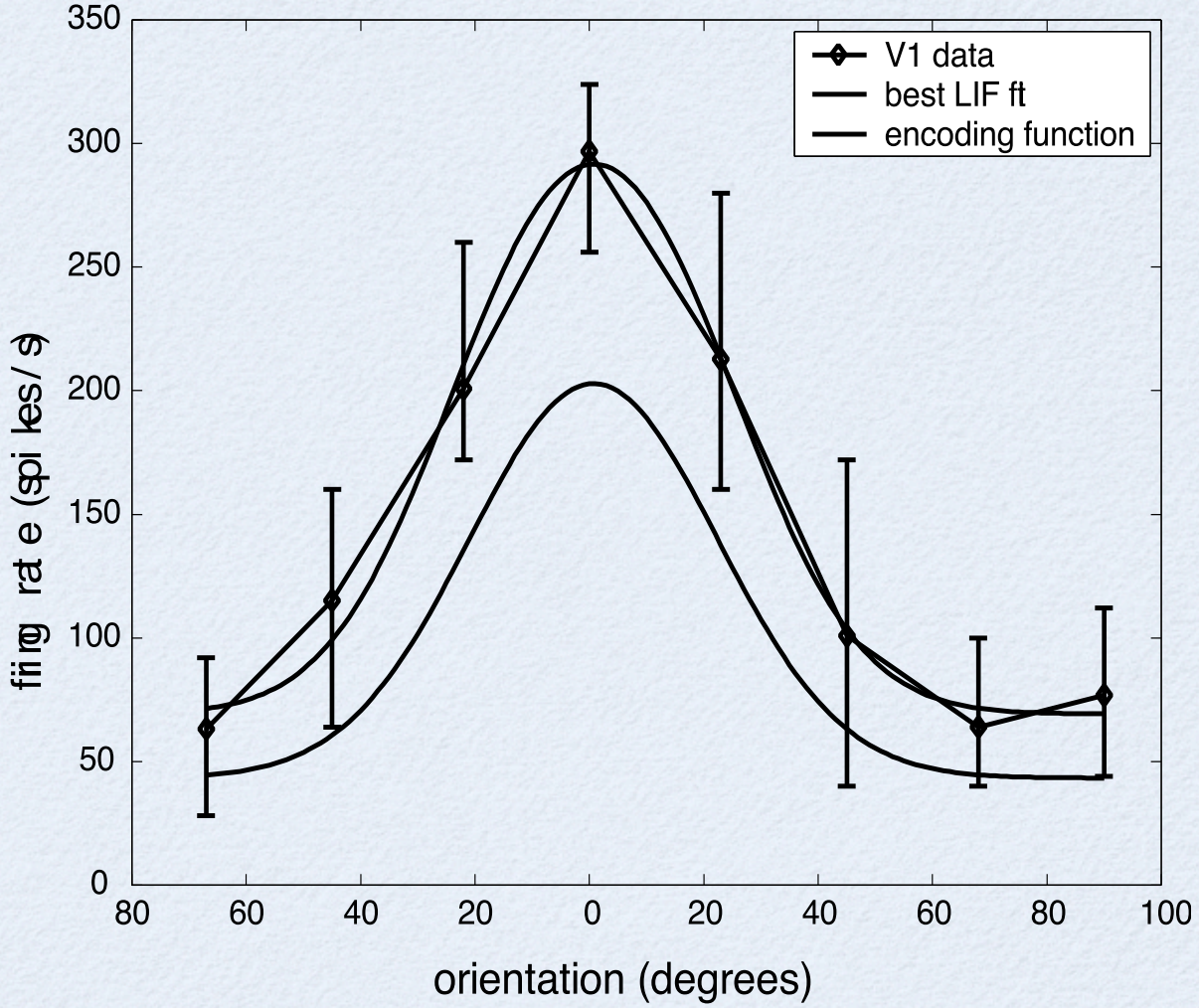
# Representational hierarchy

| *Kind* | *Encoder* ($a_i(x)$) | *Decoder* ($\hat{x}$) |
|---|---|---|
| Scalar (1) | $G_i\left[\alpha_i x + J_i^{bias}\right]$ | $\sum_i a_i(x)\phi_i$ |
| Vector ($N$) | $G_i\left[\alpha_i \left\langle \tilde{\phi}_i[n]x[n] \right\rangle_n + J_i^{bias}\right]$ | $\sum_i a_i(x[n])\phi_i[n]$ |
| Function ($\infty$) | $G_i\left[\alpha_i \left\langle \tilde{\phi}_i(\nu)x(\nu) \right\rangle_\nu + J_i^{bias}\right]$ | $\sum_i a_i(x(\nu))\phi_i(\nu)$ |
| Vector Field ($\infty \times N$) | $G_i\left[\alpha_i \left\langle \tilde{\phi}_i(\nu,[n])x(\nu,[n]) \right\rangle + J_i^{bias}\right]$ | $\sum_i a_i(x(\nu,[n])\phi_i(\nu,[n])$ |

# Why functions?

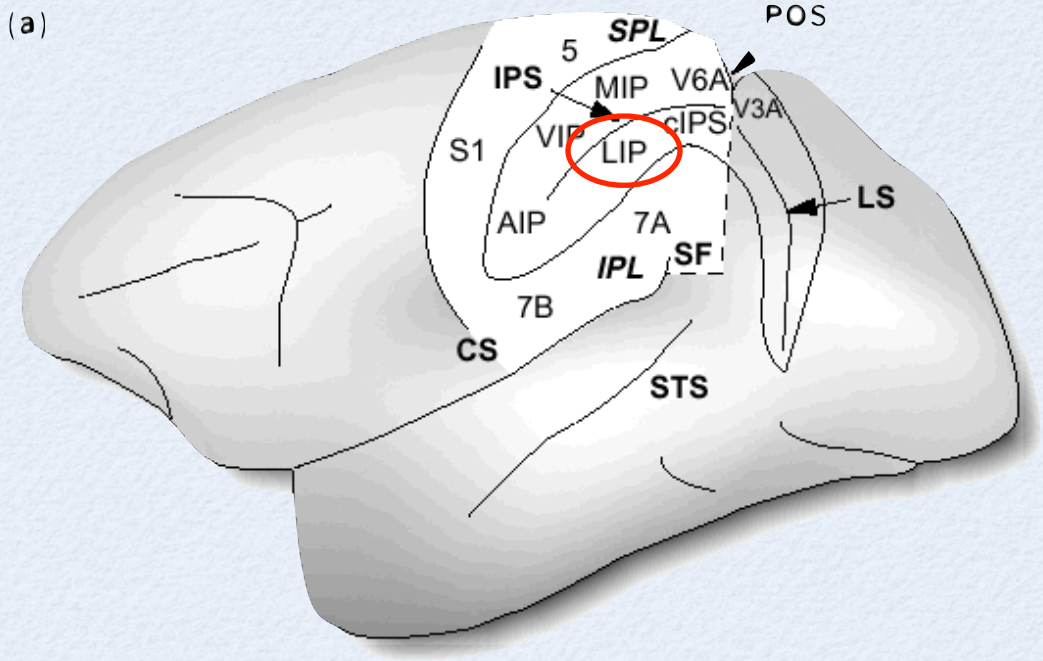
- Function representation is a common kind of representation because it is so general
- If the relation between two continuous variables (e.g. light intensity and space, pitch and time, etc.) is represented, then function representation is likely most appropriate.
- Consider primary visual cortex tuning to orientation (i.e. space by orientation)

# V1 typical tuning curve



# An extended example (LIP)

- Consider lateral intraparietal cortex (LIP).




- Evidence that differently-shaped objects at the same location cause different firing patterns

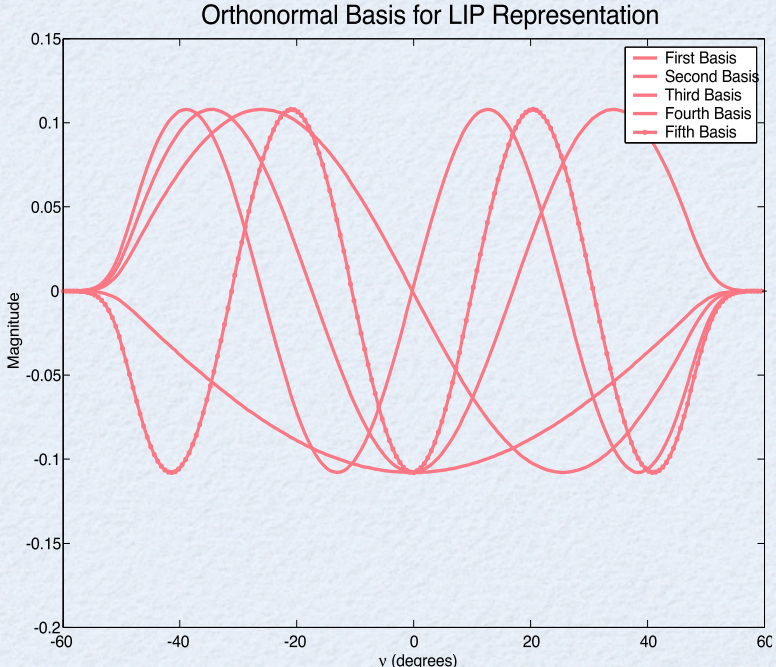
# Function representation

- Let ν be space, and $y$ be the 'other' variable (e.g. shape), then we can write $y=x(\nu)$ as the repn
- Need to specify the domain of the repn
- Do as for temporal decoding, a basis decomposition

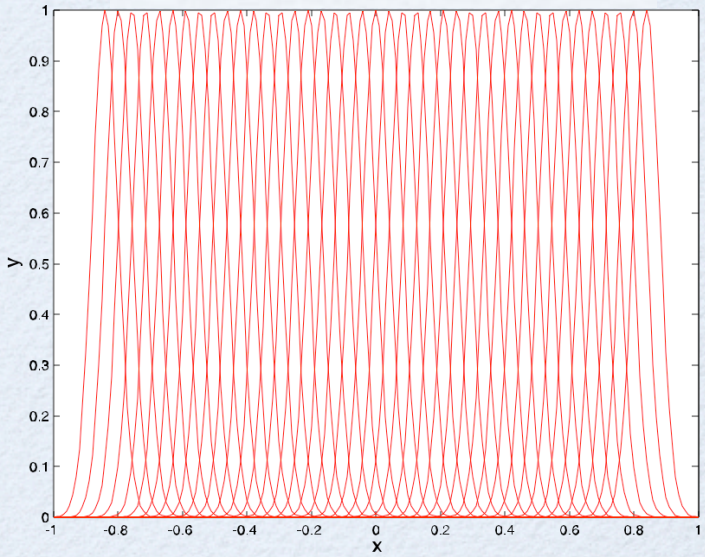
$$x(\nu; \mathbf{A}) = \sum_m A_m \Phi_m(\nu) \text{ for } \mathbf{A} \in \rho(\mathbf{A})$$

# Function representation

- LIP represents multi-bumped, variable height 'sum of Gaussian' functions.
- Find an orthogonal basis that spans the space of possible representations, $x(\nu)$.




= SVD of



# Defining the domain

- Defining $\rho(\mathbf{A})$ is a way of limiting the space spanned by $\Phi$ to a subspace of interest
- Like a Fourier representation, defining $\rho(\mathbf{A})$ gets you high/low/band-pass functions.
- It's often difficult to define the probability distribution $\rho(\mathbf{A})$.
- Instead, identify vectors $\mathbf{A}$ that represent these functions then use them as a MC sample

# Function representation

- Encoding (guesses?)

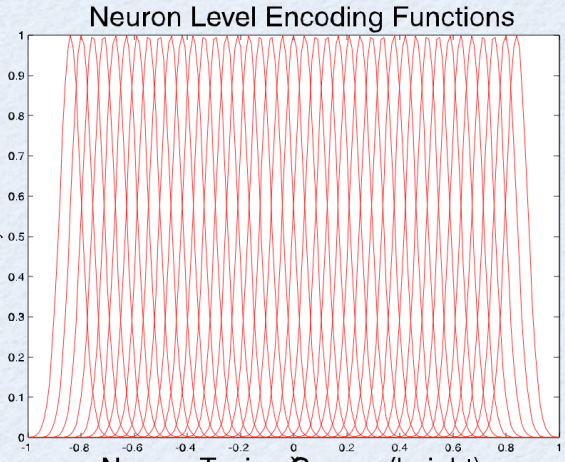
$$a_i(x(\nu;\mathbf{A})) = G_i\left[\alpha_i\left\langle x(\nu;\mathbf{A})\tilde{\phi}_i(\nu)\right\rangle_\nu + J_i^{bias}\right]$$
$$= a_i(\mathbf{A})$$

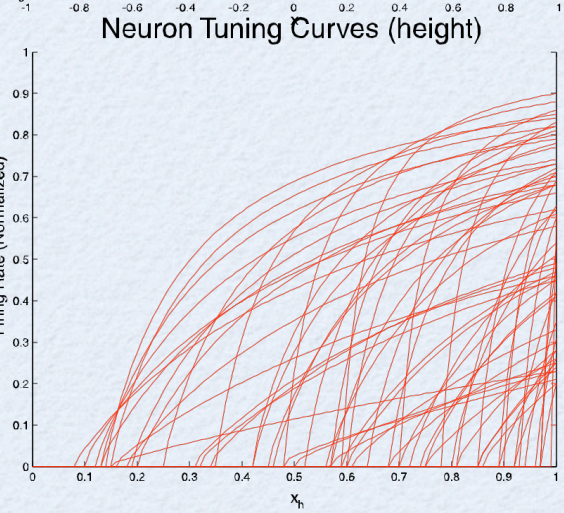
- Decoding

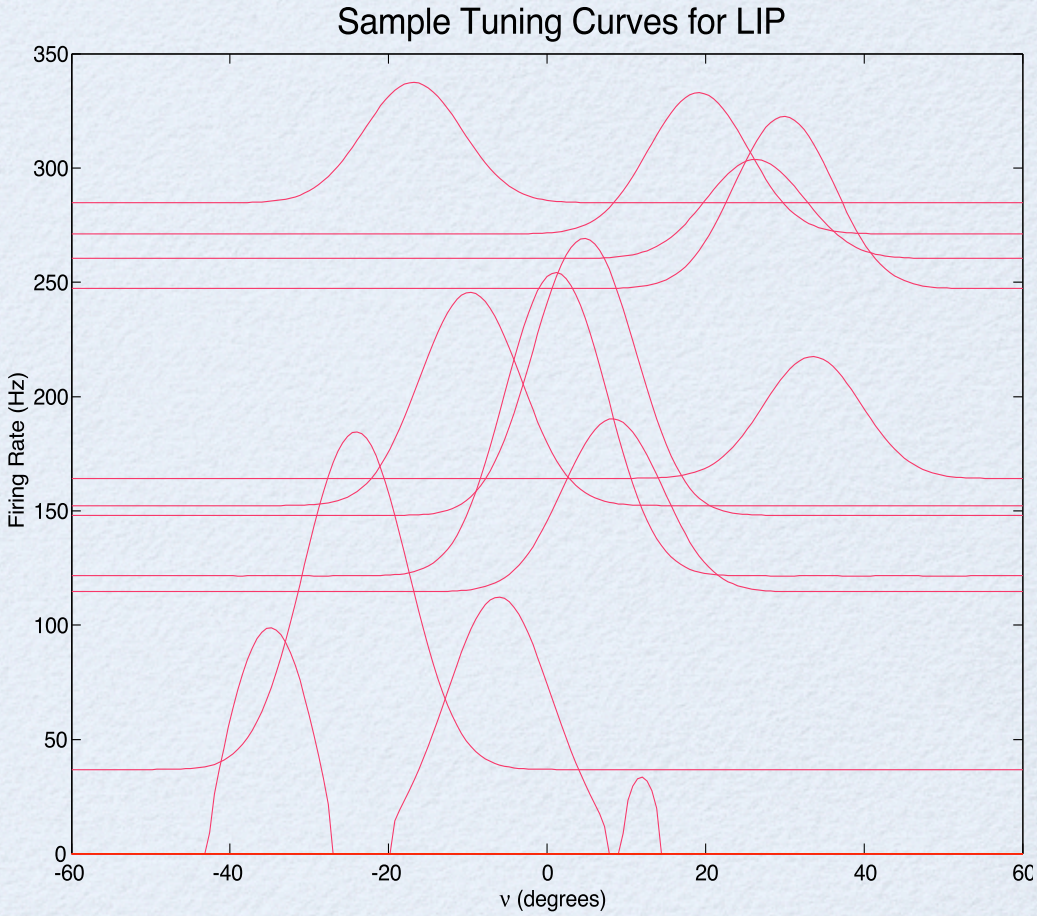
$$\hat{x}(\nu;\mathbf{A}) = \sum_i a_i(\mathbf{A})\phi_i(\nu)$$

# Function representation

- Neuron tuning properties:



# Finding decoders

- Just like before

$$E = \left\langle \left[ x(\nu;\mathbf{A}) - \sum_i \left(a_i(\mathbf{A}) + \eta_i\right)\phi_i(\nu) \right]^2 \right\rangle_{\mathbf{A},\eta}$$

- So, $\boldsymbol{\phi}(\nu) = \boldsymbol{\Gamma}^{-1}\boldsymbol{\Upsilon}(\nu)$
- Where

$$\begin{aligned} \Gamma_{ij} &= \langle a_i(\mathbf{A})a_j(\mathbf{A})\rangle_{\mathbf{A}} + \sigma_\eta^2\delta_{ij} \\ \Upsilon_i(\nu) &= \langle x(\nu;\mathbf{A})a_i(\mathbf{A})\rangle_{\mathbf{A}} \end{aligned}$$

# Consequences

- The `tuning curve' measured by neuroscientists will change depending on the input (!)
  - must be very prudent in our choice of `test' functions for the `real' tuning curve of a cell (an appropriate delta function)
- Can show the cell everything you can (white noise), and narrow down the elements of the stimulus set that the cell actually responds to

# Functions as vectors

- Useful translation:

$$\tilde{\phi}_i(\nu) = \sum_m^M \tilde{q}_{im}\Phi_m(\nu) \qquad \phi_i(\nu) = \sum_m^M q_{im}\Phi_m(\nu)$$

$$\begin{aligned}
a_i(\mathbf{A}) &= G_i\left[\alpha_i\left\langle\sum_{n,m} A_m\Phi_m(\nu)\tilde{q}_{in}\Phi_n(\nu)\right\rangle_\nu + J_i^{bias}\right] \\
&= G_i\left[\alpha_i\left(\sum_{n,m} A_m\tilde{q}_{in}\delta_{nm}\right) + J_i^{bias}\right] \\
&= G_i\left[\alpha_i\left(\sum_m A_m\tilde{q}_{im}\right) + J_i^{bias}\right] \\
&= G_i\left[\alpha_i\left\langle\mathbf{A}\tilde{\mathbf{q}}_i\right\rangle_m + J_i^{bias}\right]
\end{aligned}$$

# Functions as vectors

- So, we can express function decoding as one of vector decoding:

$$\hat{A}_m = \sum_i a_i(\mathbf{A}) q_{im}$$

$$\hat{\mathbf{A}} = \sum_i a_i(\mathbf{A}) \mathbf{q}_i$$

- Why bother with function representation?
  - 1. Some neural systems are more naturally thought of this way (reln of continuous values)
  - 2. Clarifies how different dimensions of a vector can have distinct mappings to the brain.

# What is representation

- Representation = {encoding, decoding}
- Therefore, you don't know what a representation is until you know how it's used (big phil debate).
- So, merely carrying information isn't enough to represent (e.g., bumps on the head).
- So, let's think more about decoders

# What is representation

- First, there are more than one
  $\phi_i^{\mathbf{x}}$ and $\phi_i^{f(\mathbf{x})}$ and, of course $x = f^{-1}(f(x))$
- So, let's stipulate that the representational decoders are those which identify the variable that all other actual decodings are functions of
- Still doesn't solve the inverse issue:
  - Let's appeal to consistency with other sciences (hence we'll choose variables like 'velocity')