**SYDE 556/750**

# Simulating Neurobiological Systems
# Lecture 3 and 4: Population Representation

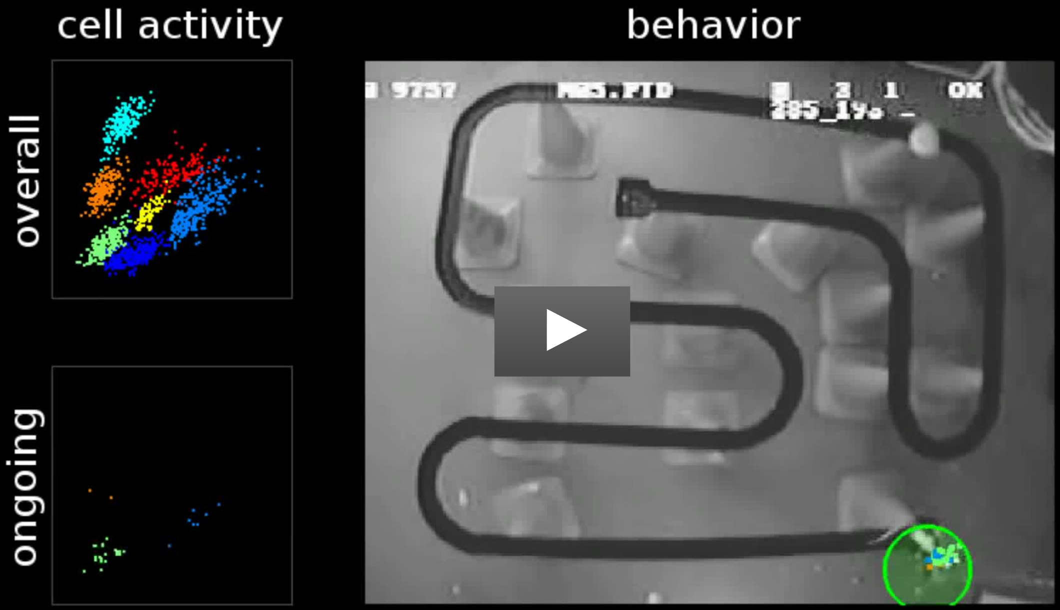
Terry Stewart

September 15 & 20, 2021

- Slide design: Andreas Stöckel
- Content: Terry Stewart, Andreas Stöckel, Chris Eliasmith

UNIVERSITY OF WATERLOO | FACULTY OF ENGINEERING

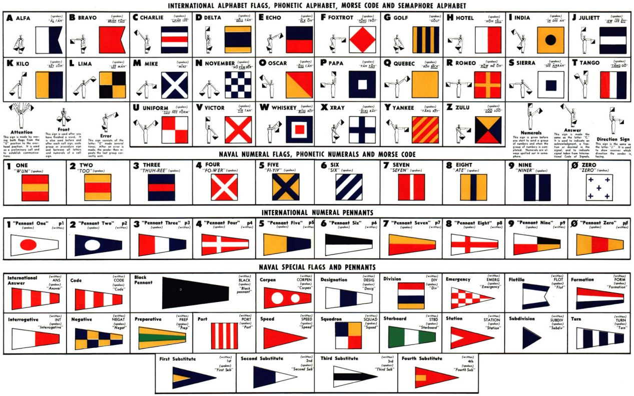
# Visual Cortex
## Mapping receptive fields


cell activity
behavior
overall
ongoing

# NEF Principle 1: Representation

**NEF Principle 1 – Representation**
*Groups* ("populations", or "ensembles") of neurons *represent* represent values via nonlinear encoding and linear decoding.

# Lossless Codes




| | | | |
|---|---|---|---|
| A | • — | U | • • — |
| B | — • • • | V | • • • — |
| C | — • — • | W | • — — |
| D | — • • | X | — • • — |
| E | • | Y | — • — — |
| F | • • — • | Z | — — • • |
| G | — — • | | |
| H | • • • • | | |
| I | • • | | |
| J | • — — — | | |
| K | — • — | 1 | • — — — — |
| L | • — • • | 2 | • • — — — |
| M | — — | 3 | • • • — — |
| N | — • | 4 | • • • • — |
| O | — — — | 5 | • • • • • |
| P | • — — • | 6 | — • • • • |
| Q | — — • — | 7 | — — • • • |
| R | • — • | 8 | — — — • • |
| S | • • • | 9 | — — — — • |
| T | — | 0 | — — — — — |

Encoding: $\mathbf{a} = f(\mathbf{x})$ Decoding: $\mathbf{x} = f^{-1}(\mathbf{a})$

# Binary numbers: Nonlinear encoding, linear decoding

- Represent a natural number between $0$ and $2^n - 1$ as $n$ binary digits.

# Binary numbers: Nonlinear encoding, linear decoding

- Represent a natural number between $0$ and $2^n - 1$ as $n$ binary digits.
- **Nonlinear encoding**

$$a_i = \big(f(x)\big)_i = \begin{cases} 1 & \text{if } x - 2^i \left\lfloor \frac{x}{2^i} \right\rfloor > 2^{i-1}\,, \\ 0 & \text{otherwise}\,. \end{cases}$$

# Binary numbers: Nonlinear encoding, linear decoding

- Represent a natural number between $0$ and $2^n - 1$ as $n$ binary digits.
- **Nonlinear encoding**

$$a_i = \big(f(x)\big)_i = \begin{cases} 1 & \text{if } x - 2^i \left\lfloor \frac{x}{2^i} \right\rfloor > 2^{i-1}, \\ 0 & \text{otherwise}. \end{cases}$$

- **Linear decoding**

$$x = f^{-1}(\mathbf{a}) = \sum_{i=0}^{n-1} 2^i a_i = \mathbf{F}\mathbf{a} = \begin{pmatrix} 1 & 2 & \dots & 2^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix}.$$

# Binary numbers: Nonlinear encoding, linear decoding

- Represent a natural number between $0$ and $2^n - 1$ as $n$ binary digits.
- **Nonlinear encoding**

$$a_i = \big(f(x)\big)_i = \begin{cases} 1 & \text{if } x - 2^i \left\lfloor \frac{x}{2^i} \right\rfloor > 2^{i-1}\,, \\ 0 & \text{otherwise}\,. \end{cases}$$

- **Linear decoding**

$$x = f^{-1}(\mathbf{a}) = \sum_{i=0}^{n-1} 2^i a_i = \mathbf{F}\mathbf{a} = \begin{pmatrix} 1 & 2 & \dots & 2^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix}\,.$$

- This is a **distributed code**.

# Binary numbers: Nonlinear encoding, linear decoding

- Represent a natural number between $0$ and $2^n - 1$ as $n$ binary digits.
- **Nonlinear encoding**

$$a_i = \left(f(x)\right)_i = \begin{cases} 1 & \text{if } x - 2^i \left\lfloor \frac{x}{2^i} \right\rfloor > 2^{i-1}\,, \\ 0 & \text{otherwise}\,. \end{cases}$$

- **Linear decoding**

$$x = f^{-1}(\mathbf{a}) = \sum_{i=0}^{n-1} 2^i a_i = \mathbf{F}\mathbf{a} = \begin{pmatrix} 1 & 2 & \dots & 2^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix}\,.$$

- This is a **distributed code**. But, **not robust** against additive noise!

# Lossy codes

- **Lossy code**
  Inverse $f^{-1}$ does not exist, instead *approximate* the represented value

$$\text{Encoding:}\quad \mathbf{a} = f(\mathbf{x}) \qquad\qquad \text{Decoding:}\quad \mathbf{x} \approx g(\mathbf{a})$$

# Lossy codes

- **Lossy code**
  Inverse $f^{-1}$ does not exist, instead *approximate* the represented value

  $$\text{Encoding:} \quad \mathbf{a} = f(\mathbf{x}) \qquad\qquad \text{Decoding:} \quad \mathbf{x} \approx g(\mathbf{a})$$

- **Examples**
  - Audio, image, and video coding schemes
    (MP3, JPEG, H.264)
  - Basis transformation onto first $n$ principal components (PCA)

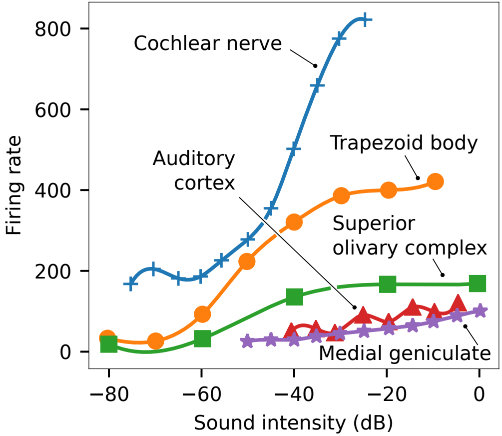
# Lossy codes

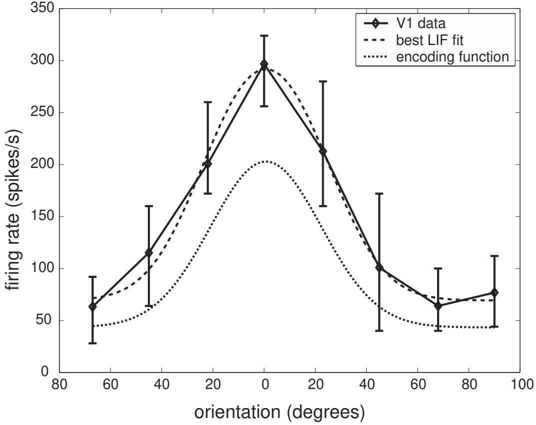
- **Lossy code**
  Inverse $f^{-1}$ does not exist, instead *approximate* the represented value

$$\text{Encoding:} \quad \mathbf{a} = f(\mathbf{x}) \qquad\qquad \text{Decoding:} \quad \mathbf{x} \approx g(\mathbf{a})$$

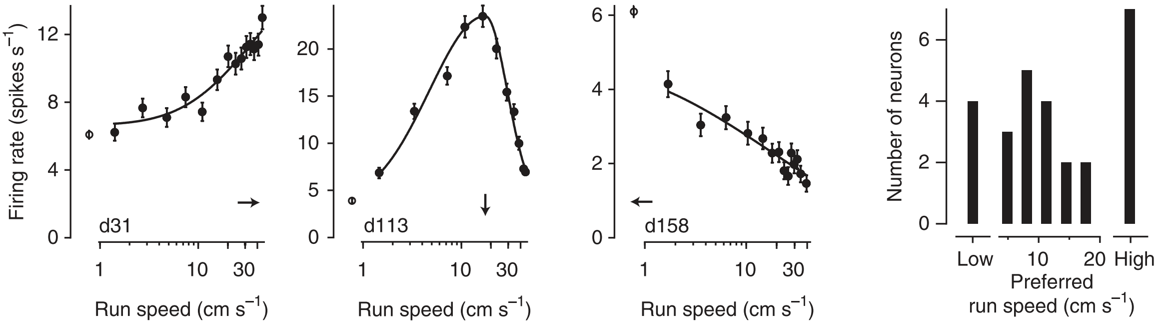
- **Examples**
  - Audio, image, and video coding schemes (MP3, JPEG, H.264)
  - Basis transformation onto first $n$ principal components (PCA)
  - **Neural Representations**

# Tuning curves (I)

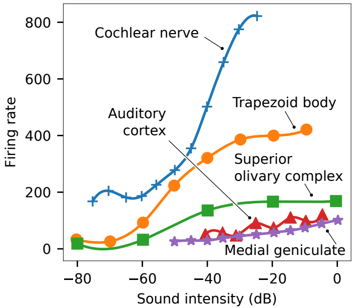


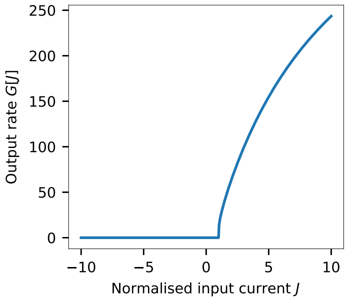



# Tuning curves (II)



- Last lecture: response curves: $a = G(J)$
- This lecture: tuning curves: $a = f(x) = G(J_i(x))$
- What sort of function can we try for $J_i(x)$?

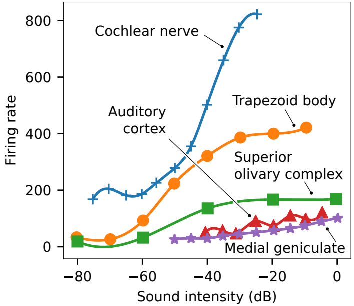


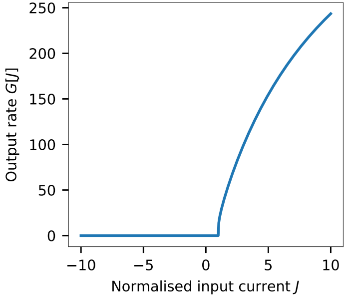



- Last lecture: response curves: $a = G(J)$
- This lecture: tuning curves: $a = f(x) = G(J_i(x))$
- What sort of function can we try for $J_i(x)$?
- Introduce a gain $\alpha_i$ and a bias $J_i^{\mathrm{bias}}$:

$$J_i(x) = \alpha_i x + J_i^{\mathrm{bias}}$$
$$a_i(x) = G(\alpha_i x + J_i^{\mathrm{bias}})$$

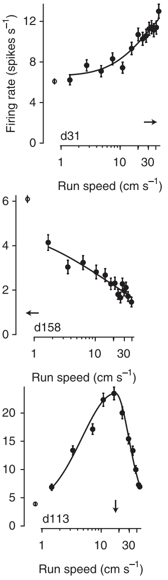
- $\alpha_i$ controls the slope
- $J_i^{\mathrm{bias}}$ shifts curve left and right



- Does this work for all tuning curves?

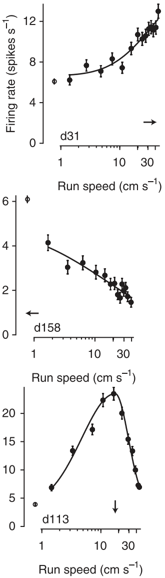
$$a_i(x) = G(\alpha_i x + J_i^{\text{bias}})$$



- Does this work for all tuning curves?

$$a_i(x) = G(\alpha_i x + J_i^{\text{bias}})$$

- a) increasing: Yes!

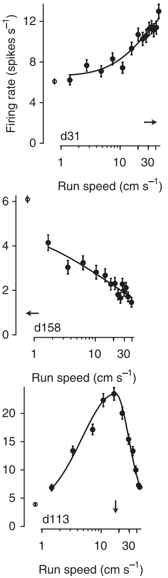


- Does this work for all tuning curves?

$$a_i(x) = G(\alpha_i x + J_i^{\text{bias}})$$

- a) increasing: Yes!
- b) decreasing: Yes! (just let $\alpha_i$ be negative)
  - or, better yet, introduce $e_i$ which is either 1 or -1 and keep $\alpha_i$ to be always positive. This keeps the two ideas (slope and increase/decreasing) separate.

$$a_i(x) = G(\alpha_i (e_i x) + J_i^{\text{bias}})$$



- Does this work for all tuning curves?

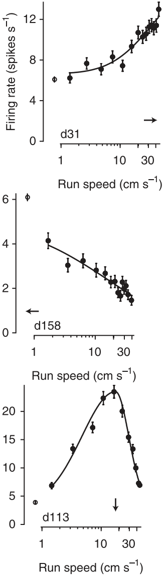
$$a_i(x) = G(\alpha_i x + J_i^{\text{bias}})$$

- a) increasing: Yes!
- b) decreasing: Yes! (just let $\alpha_i$ be negative)
  - or, better yet, introduce $e_i$ which is either 1 or -1 and keep $\alpha_i$ to be always positive. This keeps the two ideas (slope and increase/decreasing) separate.

$$a_i(x) = G(\alpha_i (e_i x) + J_i^{\text{bias}})$$

- c) preferred stimulus: Need some sort of similarity measure
  - But it shouldn't be too complicated. So far we've only needed to introduce multiplication and addition, which are both things we're pretty sure neurons can do, so let's avoid adding anything else if we don't have to. Ideas?

$$a_i(x) = G(\alpha_i sim(e_i, x) + J_i^{\text{bias}})$$



# Encoders: Preferred Direction Vectors

- The represented value $x$ doesn't have to be a scalar
- What if it's a vector?

# Encoders: Preferred Direction Vectors

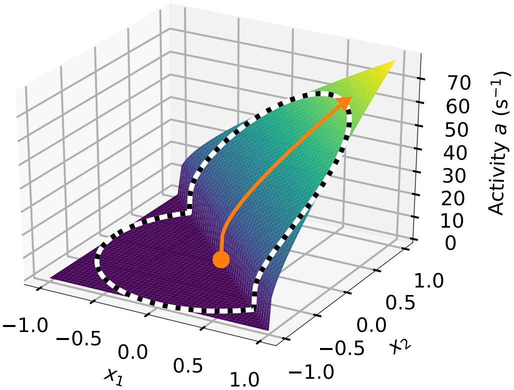
- The represented value $x$ doesn't have to be a scalar
- What if it's a vector?
- There's a simple similarity-like measure for vectors: the dot product

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=0}^{d} x_i y_i = \cos(\angle(\mathbf{x}, \mathbf{y})) \|\mathbf{x}\| \|\mathbf{y}\|$$

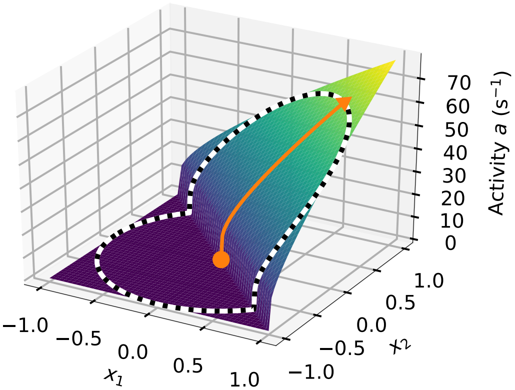
$$a_i(\mathbf{x}) = G(\alpha_i \langle \mathbf{e}_i, \mathbf{x} \rangle + J_i^{\mathrm{bias}})$$

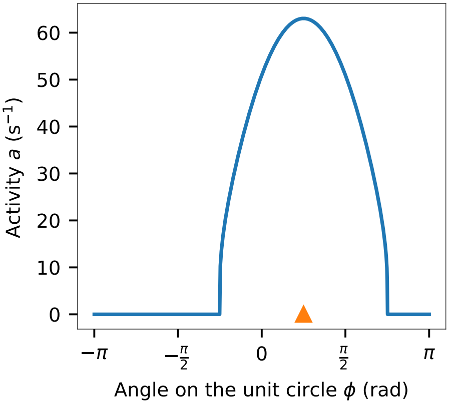
- Constrain $e_i$ to be a unit vector
  - Note that for scalar $x$, the only two unit vectors are $+1$ and -1
  - So the increasing / decreasing scenario is a special case of this!

# Preferred Directions in Higher Dimensions: Representing 2D Values



# Preferred Directions in Higher Dimensions: Representing 2D Values



# Decoding

- Non-linear Encoding and Linear Decoding

$$\mathbf{a}_i = G\left[\alpha_i \langle \mathbf{x}, \mathbf{e}_i \rangle + J_i^{\text{bias}}\right], \qquad \text{Encoding}$$
$$\hat{\mathbf{x}} = \mathbf{D}\mathbf{a} \qquad \text{Decoding}$$

- How do we find $\mathbf{D}$?

# Decoding

- Non-linear Encoding and Linear Decoding

$$\mathbf{a}_i = G\left[\alpha_i \langle \mathbf{x}, \mathbf{e}_i \rangle + J_i^{\text{bias}}\right], \qquad \text{Encoding}$$

$$\hat{\mathbf{x}} = \mathbf{D}\mathbf{a} \qquad \text{Decoding}$$

- How do we find $\mathbf{D}$?
- Least-squares minimization

$$\arg\min_{\mathbf{D}} E = \frac{1}{|\mathbb{X}|} \int_{\mathbb{X}} \|\mathbf{x} - \hat{\mathbf{x}}\| \, \mathrm{d}\mathbf{x} = \frac{1}{|\mathbb{X}|} \int_{\mathbb{X}} \|\mathbf{x} - \mathbf{D}\mathbf{a}(\mathbf{x})\| \, \mathrm{d}\mathbf{x}$$

# Decoding via Least-squares Minimization

- Find the minimum decoding error

$$\arg\min_{\mathbf{D}} E = \frac{1}{|\mathbb{X}|} \int_{\mathbb{X}} \|\mathbf{x} - \hat{\mathbf{x}}\| \, \mathrm{d}\mathbf{x} = \frac{1}{|\mathbb{X}|} \int_{\mathbb{X}} \|\mathbf{x} - \mathbf{D}\mathbf{a}(\mathbf{x})\| \, \mathrm{d}\mathbf{x}$$

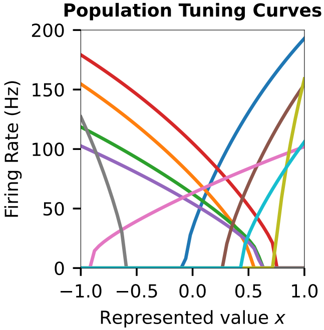
- Can't do that analytically (in general), so let's sample

$$\arg\min_{\mathbf{D}} E = \frac{1}{N} \sum_{i=0}^{N} \|\mathbf{x}_i - \mathbf{D}\mathbf{a}(\mathbf{x}_i)\|$$

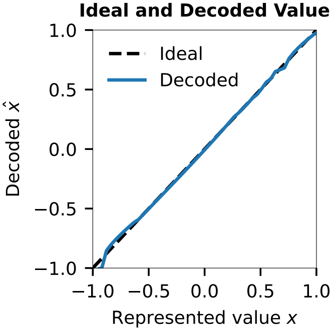
# Decoding via Least-squares Minimization

- Let's write this in matrix form, where $\mathbf{A}_{ik} = a_i(x_k)$ and $\mathbf{X} = (x_1, \ldots, x_N)$
- We want $\mathbf{A}^T\mathbf{D}^T = \mathbf{X}^T$
- So $\mathbf{A}\mathbf{A}^T\mathbf{D}^T = \mathbf{A}\mathbf{X}^T$
- $(\mathbf{A}\mathbf{A}^T)^{-1}\mathbf{A}\mathbf{A}^T\mathbf{D}^T = (\mathbf{A}\mathbf{A}^T)^{-1}\mathbf{A}\mathbf{X}^T$
- $\mathbf{D}^T = (\mathbf{A}\mathbf{A}^T)^{-1}\mathbf{A}\mathbf{X}^T$
- In Python, `D = np.linalg.lstsq(A.T, X.T, rcond=None)[0].T`
- (where `A` is a n x N array and `X` is a d x N array)

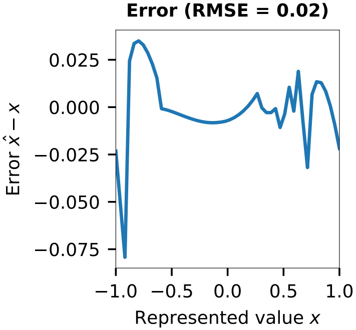
# Decoding




$\mathbf{A}$



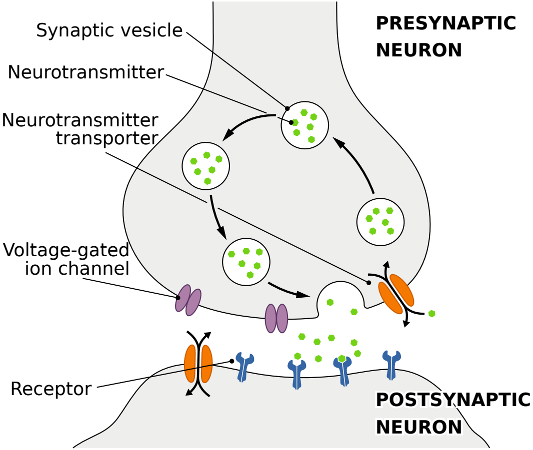

$\mathbf{A}^T\mathbf{D}^T$




$\mathbf{A}^T\mathbf{D}^T - \mathbf{X}^T$

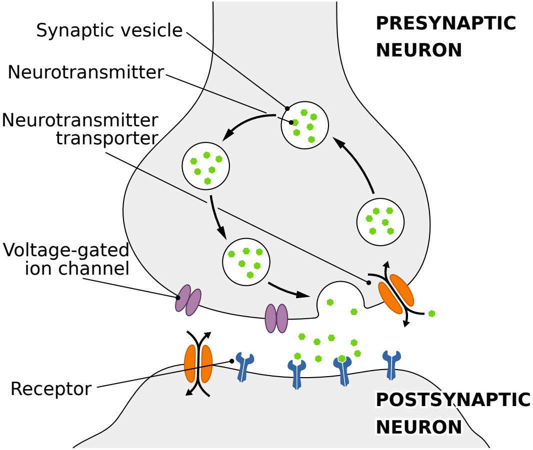
# Sources of Noise in Biological Neural Networks

- **Axonal jitter**
  Active axonal spike propagation
- **Vesicle release failure**
  10-30% of pre-synaptic events cause post-synaptic current
- **Neurotransmitter per vesicle**
  Varying amounts of neurotransmitter
- **Ion channel noise**
  Ion-channels are "binary", stochastic
- **Thermal noise**
- **Network effects**
  Simple, noise-free inhibitory/excitatory networks produce irregular spike trains



# Sources of Noise in Biological Neural Networks

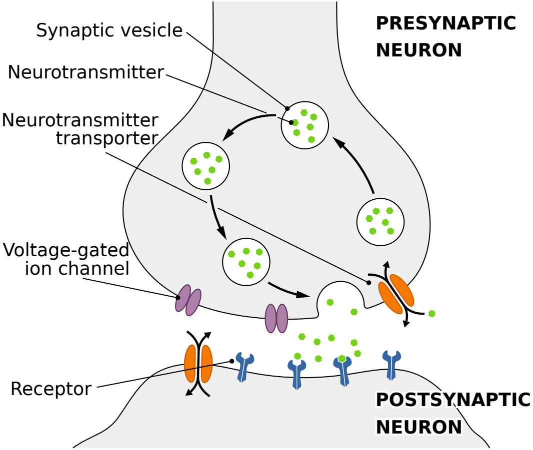
- **Axonal jitter**
  Active axonal spike propagation
- **Vesicle release failure**
  10-30% of pre-synaptic events cause post-synaptic current
- **Neurotransmitter per vesicle**
  Varying amounts of neurotransmitter
- **Ion channel noise**
  Ion-channels are “binary”, stochastic
- **Thermal noise**
- **Network effects**
  Simple, noise-free inhibitory/excitatory networks produce irregular spike trains




- **How to model?**

# Sources of Noise in Biological Neural Networks

- **Axonal jitter**
  Active axonal spike propagation
- **Vesicle release failure**
  10-30% of pre-synaptic events cause post-synaptic current
- **Neurotransmitter per vesicle**
  Varying amounts of neurotransmitter
- **Ion channel noise**
  Ion-channels are "binary", stochastic
- **Thermal noise**
- **Network effects**
  Simple, noise-free inhibitory/excitatory networks produce irregular spike trains

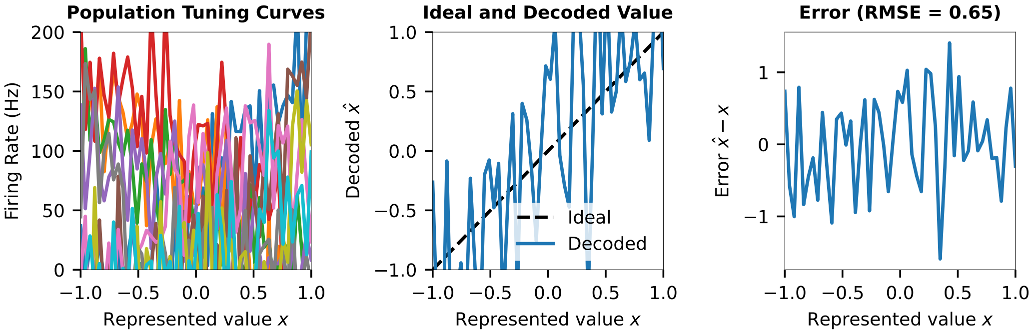



- **How to model?** Gaussian noise

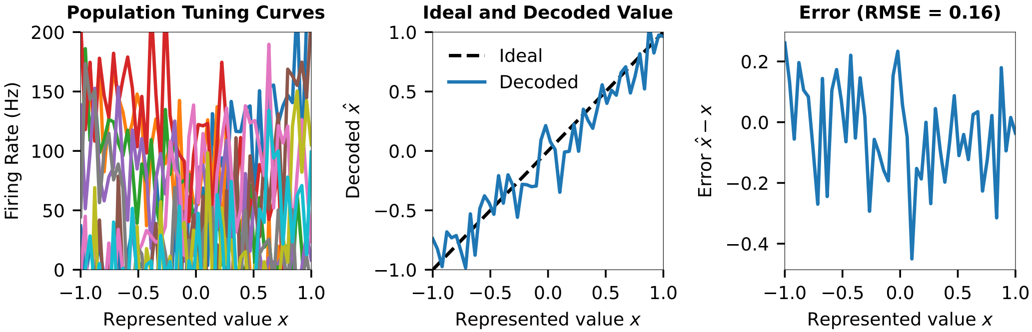
# NEF Principle 0: Noise

**NEF Principle 0 – Noise**
Biological neural systems are subject to significant amounts of noise from various sources. Any analysis of such systems must take the effects of noise into account.

# Decoding Noisy $\mathbf{A}$ Without Taking Noise Into Account



# Decoding Noisy $\mathbf{A}$ Accounting for Noise

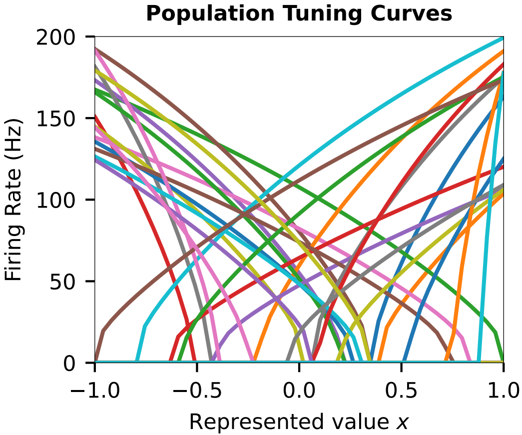


# Summary: Building a model of neural representation (Encoding)

**Encoding**

- Select $d$, possible range $\mathbf{x} \in \mathbb{X}$, usually $\mathbb{X} = \left\{\mathbf{x} \mid \|\mathbf{x}\| \leq r, \mathbf{x} \in \mathbb{R}^d\right\}$ $(r = 1)$
- Select number of neurons $n$
- Select tuning curves, maximum rates $\Rightarrow \mathbf{e}_i,\ \alpha_i,\ J_i^{\mathrm{bias}}$
  - Sample $\mathbf{e}_i$ from unit-sphere
  - Uniformly distribute $x$-intercept, maximum rate
- Encoding equation:

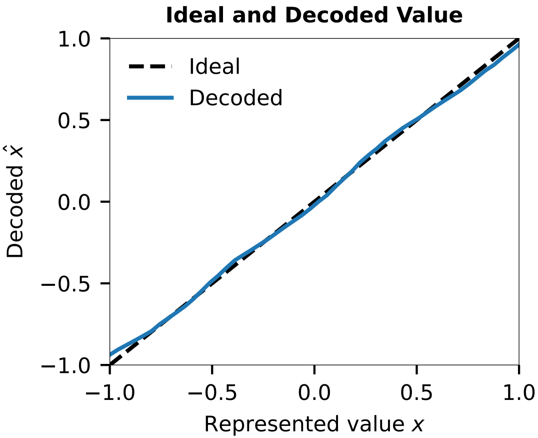
$$a_i(\mathbf{x}) = G\left[\alpha_i \langle \mathbf{e}_i, \mathbf{x} \rangle + J_i^{\mathrm{bias}}\right]$$



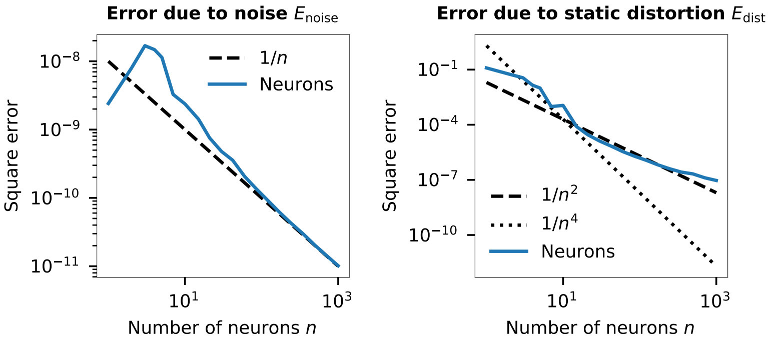
# Summary: Building a model of neural representation (Decoding)

## Decoding

- Uniformly sample $N$ samples from $\mathbb{X}$, $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$
- Compute $\mathbf{A}$, where $(\mathbf{A})_{ik} = a_i(\mathbf{x}_k)$
- Decoder computation:
$$\mathbf{D}^\mathsf{T} = \left(\mathbf{A}\mathbf{A}^\mathsf{T} + N\sigma^2\mathbf{I}\right)^{-1}\mathbf{A}\mathbf{X}^\mathsf{T}$$
- Decoding equation:
$$\hat{\mathbf{X}} = \mathbf{D}\mathbf{A}$$

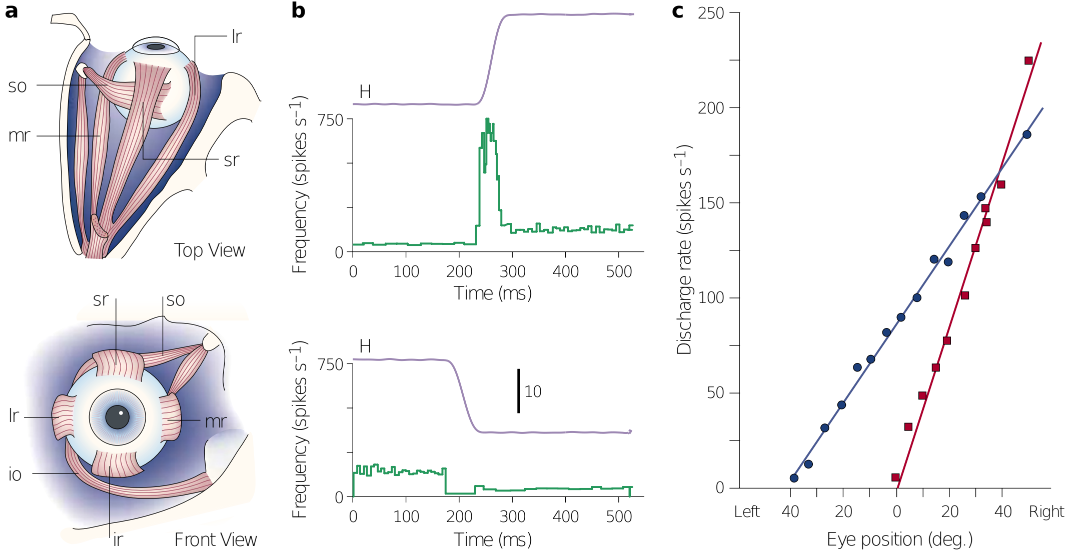


# Analysing Sources of Errors




$$E = \underbrace{\frac{1}{2}\int_{-1}^{1}\left(x - \sum_{i=1}^{n} d_i a_i(x)\right)^2 \mathrm{d}x}_{E_{\mathrm{dist}}} + \underbrace{\frac{1}{2}\sigma^2 \sum_{i=1}^{n} d_i^2}_{E_{\mathrm{noise}}}$$

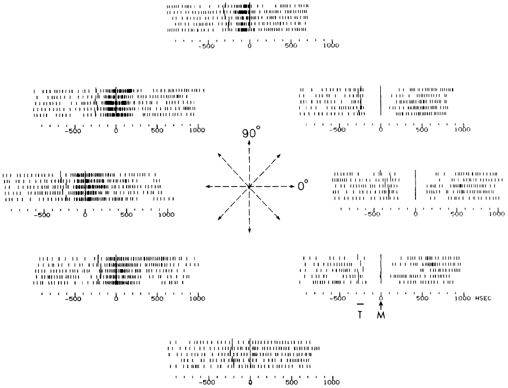
# Example: Horizontal Eye Position (1D)

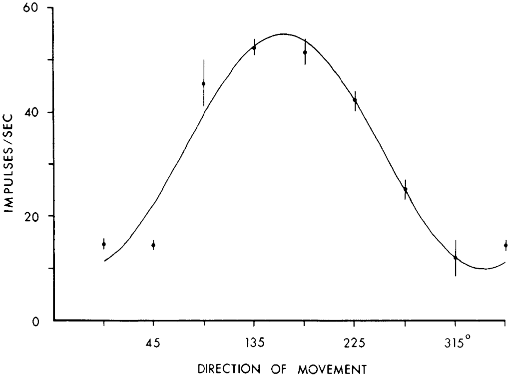


# Example: Horizontal Eye Position (1D) (cont.)

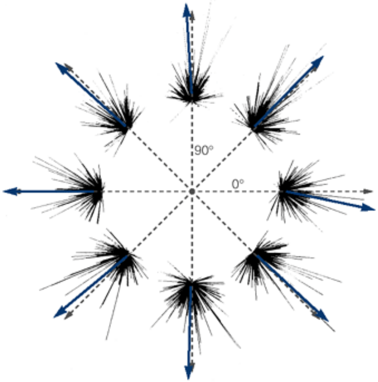
- **Step 1: System Description**
  - What is being represented?
    - $x$ is the horizontal eye position
  - What is the tuning curve shape?
    - Linear, low $\tau_{\mathrm{ref}}$, high $\tau_{\mathrm{RC}}$
    - $e_i \in \{1, -1\}$
    - Firing rates up to $300\,\mathrm{s}^{-1}$
- **Step 2: Design Specification**
  - Range of values
    - $\mathbb{X} = [-60, 60]$
  - Amount of noise
    - About $20\%$ of $\max(\mathbf{A})$
- **Step 3: Implementation**
  - Choose tuning curve parameters
  - Compute decoders

# Example: Arm Movements (2D)



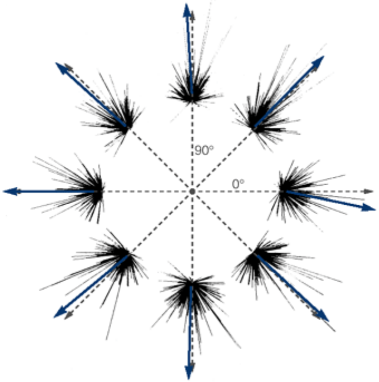
# Example: Arm Movements (2D) (cont.)




- Experiment by Georgopoulos et al., 1982
- Preferred arm movement directions $\mathbf{e}_i$
- **Idea:** *Population Vectors*, decode using

$$\hat{\mathbf{x}} = \sum_{i=1}^{n} a_i(\mathbf{x})\mathbf{e}_i = \mathbf{EA}$$

# Example: Arm Movements (2D) (cont.)

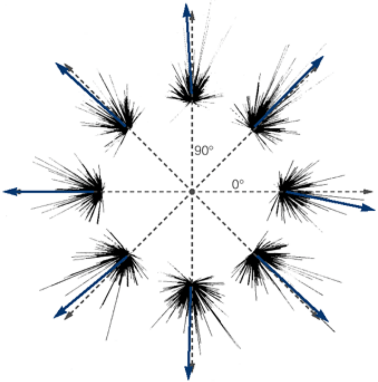



- Experiment by Georgopoulos et al., 1982
- Preferred arm movement directions $\mathbf{e}_i$
- **Idea:** *Population Vectors*, decode using

$$\hat{\mathbf{x}} = \sum_{i=1}^{n} a_i(\mathbf{x})\mathbf{e}_i = \mathbf{EA}$$

⊕ Good direction estimate

# Example: Arm Movements (2D) (cont.)



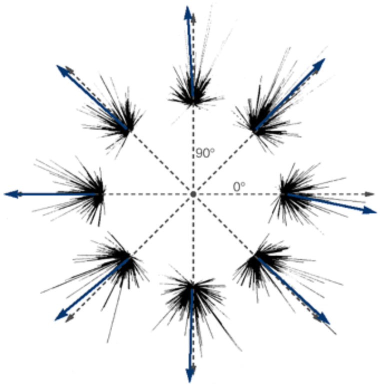

- Experiment by Georgopoulos et al., 1982
- Preferred arm movement directions $\mathbf{e}_i$
- **Idea:** *Population Vectors*, decode using

$$\hat{\mathbf{x}} = \sum_{i=1}^{n} a_i(\mathbf{x})\mathbf{e}_i = \mathbf{EA}$$

⊕ Good direction estimate

⊖ Cannot reconstruct magnitude

# Example: Arm Movements (2D) (cont.)




- Experiment by Georgopoulos et al., 1982
- Preferred arm movement directions $\mathbf{e}_i$
- **Idea:** *Population Vectors*, decode using

$$\hat{\mathbf{x}} = \sum_{i=1}^{n} a_i(\mathbf{x})\mathbf{e}_i = \mathbf{E}\mathbf{A}$$

⊕ Good direction estimate

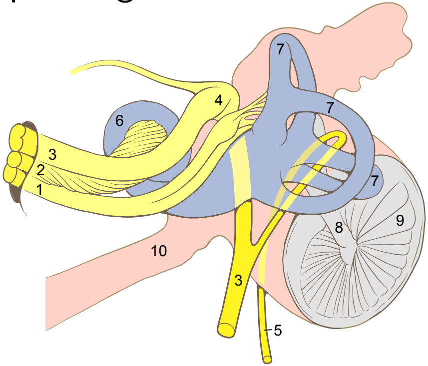
⊖ Cannot reconstruct magnitude

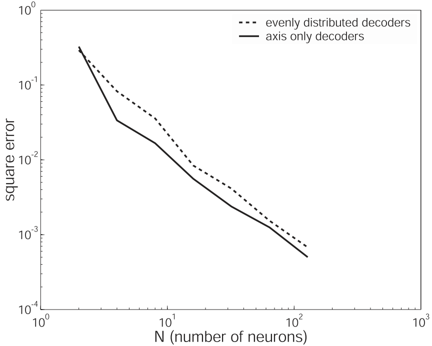
**The NEF does not use population vectors!**

# Example: Arm Movements (2D) (cont.)

- **Step 1: System Description**
  - What is being represented?
    - $\mathbf{x}$ the movement direction (or hand position)
  - What is the tuning curve shape?
    - Bell-shaped
    - Encoders are randomly distributed along the unit circle
    - Firing rates up to $60\,\mathrm{s}^{-1}$
- **Step 2: Design Specification**
  - Range of values
    - $\mathbb{X} = \{\mathbf{x} \mid \|\mathbf{x}\| \leq r, \mathbf{x} \in \mathbb{R}^2\}$
  - Amount of noise
    - About $20\%$ of $\max(\mathbf{A})$
- **Step 3: Implementation**
  - Choose tuning curve parameters
  - Compute decoders

# Example: Higher Dimensional Representation

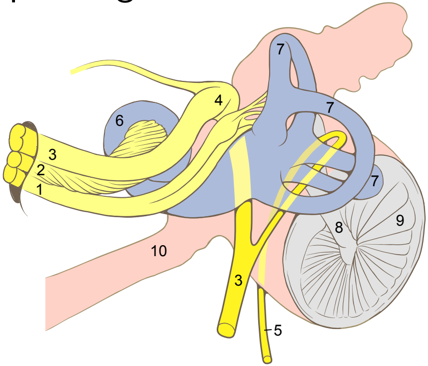


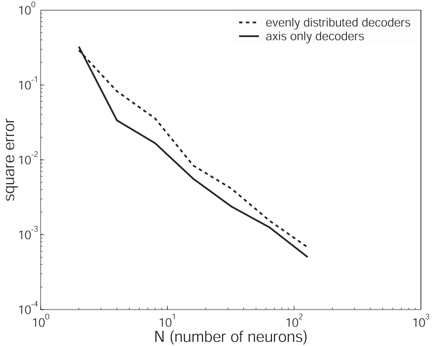




- Vestibular system senses head acceleration in 3D
- Axis aligned, must choose $\mathbf{e}_i \in \{[1,0,0],[-1,0,0],\ldots,[0,0,-1]\}$

- Same as three 1D populations
- Slightly lower precision

# Example: Higher Dimensional Representation




- Vestibular system senses head acceleration in 3D
- Axis aligned, must choose $\mathbf{e}_i \in \{[1,0,0],[-1,0,0],\dots,[0,0,-1]\}$

- Same as three 1D populations
- Slightly lower precision
- **Encoders affect accuracy**

## Administration

- **Assignment 1 has been released.**

  The due date is October 4, 2021.

# Image sources

**Title slide**
"The Ultimate painting."
Author: Clark Richert.
From Wikimedia.