SYDE 556/750

Simulating Neurobiological Systems
Lecture 5: Feed-Forward Transformation

Chris Eliasmith

September 27 & October 2, 2023

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

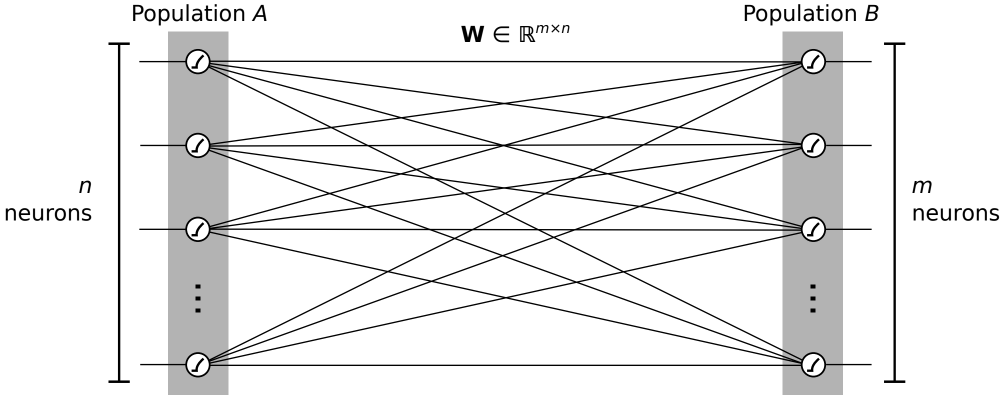
UNIVERSITY OF WATERLOO | FACULTY OF ENGINEERING

Introduction

- We've only talked about representation til now
  - What about computation?
- We start by focusing on the state of a network after learning and development
- A kind of hypothesis testing and generation

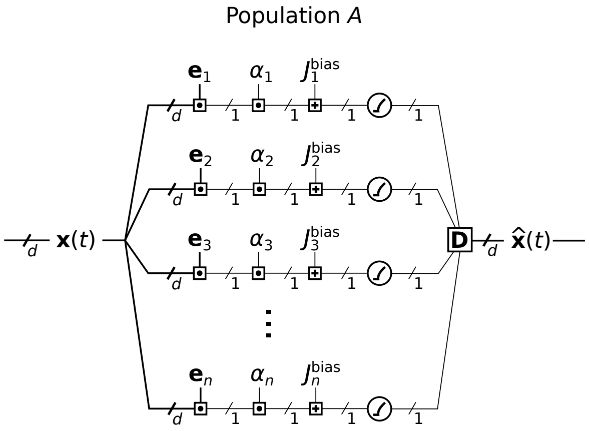
DALL-E AI Generated Art, 2022

NEF Principle 2: Transformation

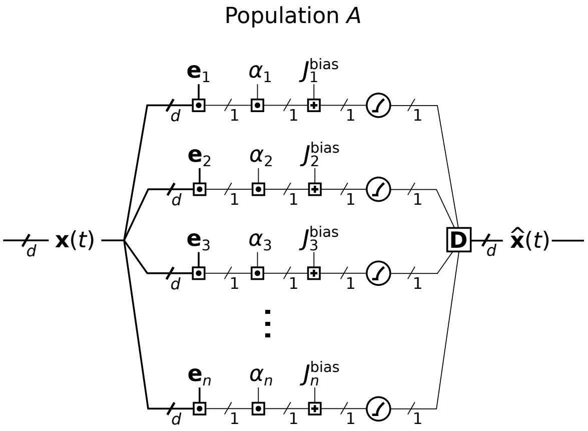


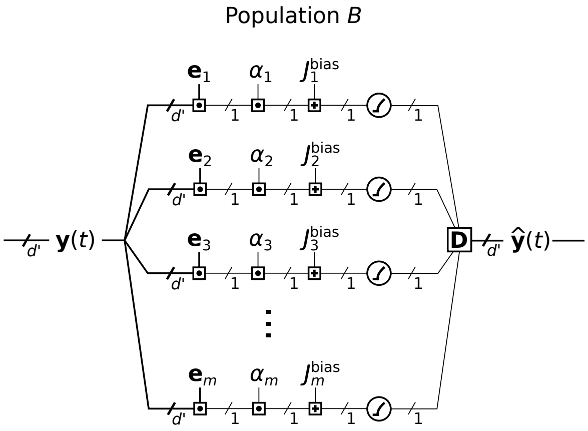
NEF Principle 2 – Transformation
Connections between populations describe *transformations* of neural representations. Transformations are functions of the variables represented by neural populations.

A Tale of Two Populations (I)

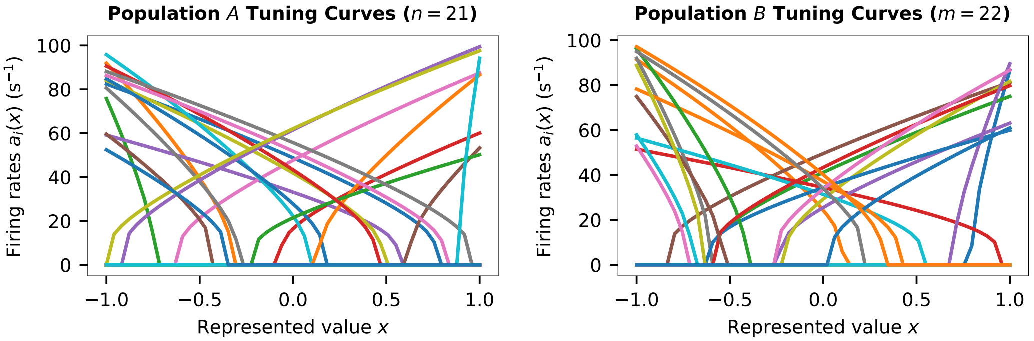


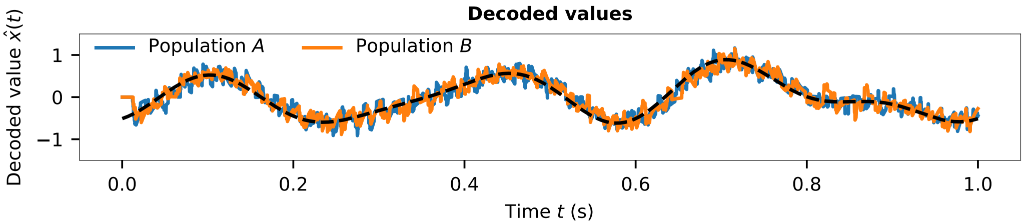
A Tale of Two Populations (I)


Population A
$\mathbf{x}(t)$
d
$\mathbf{e}_1$ α_1 J_1^{bias}
$\mathbf{e}_2$ α_2 J_2^{bias}
$\mathbf{e}_3$ α_3 J_3^{bias}
$\mathbf{e}_n$ α_n J_n^{bias}
1
D
$\hat{\mathbf{x}}(t)$

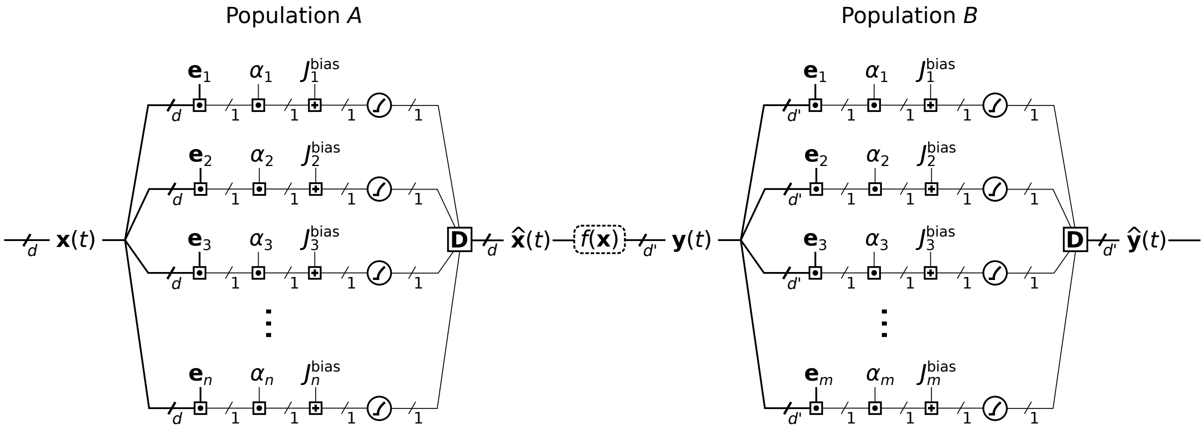

Population B
$\mathbf{y}(t)$
d'
$\mathbf{e}_1$ α_1 J_1^{bias}
$\mathbf{e}_2$ α_2 J_2^{bias}
$\mathbf{e}_3$ α_3 J_3^{bias}
$\mathbf{e}_m$ α_m J_m^{bias}
1
D
$\hat{\mathbf{y}}(t)$

Communication Channel Experiment: Same input signal

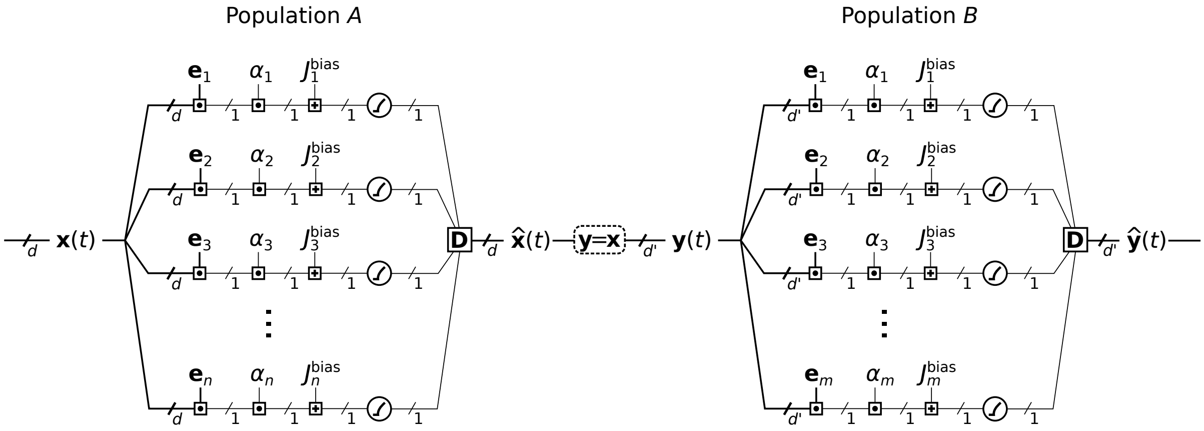




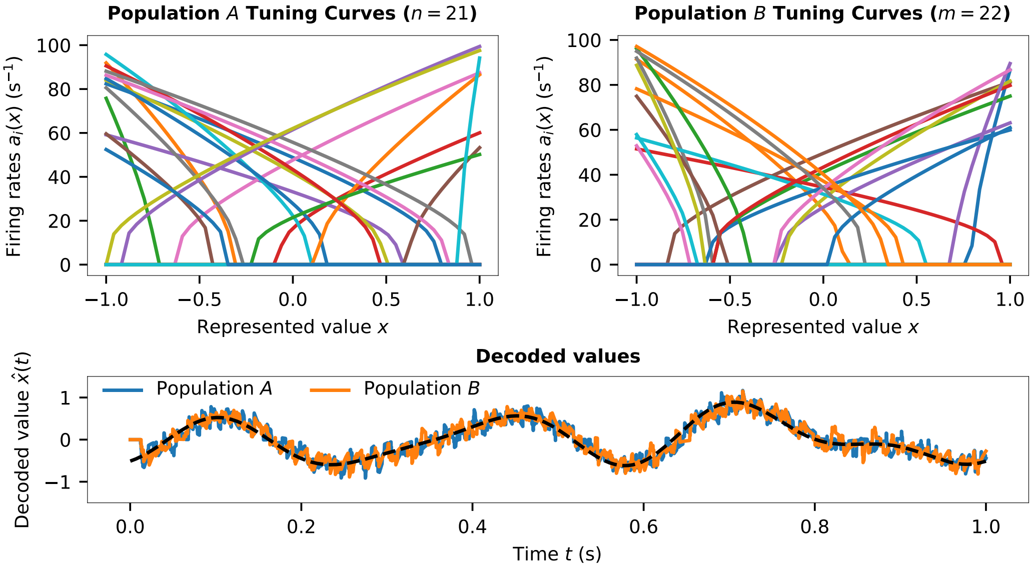
A Tale of Two Populations (II)



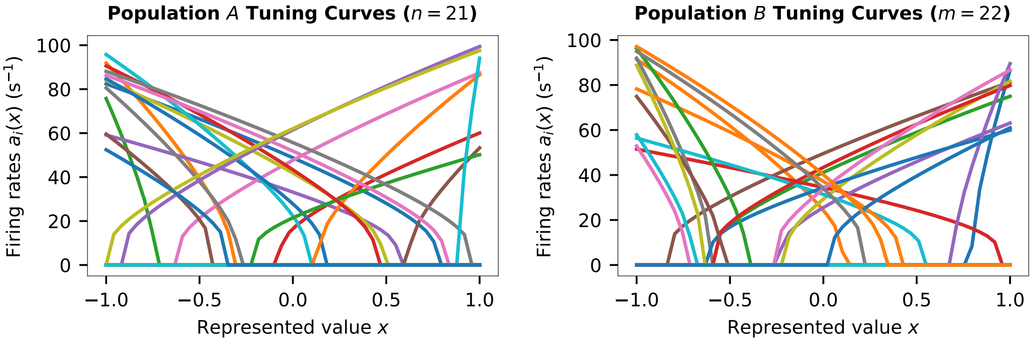
A Tale of Two Populations (II)

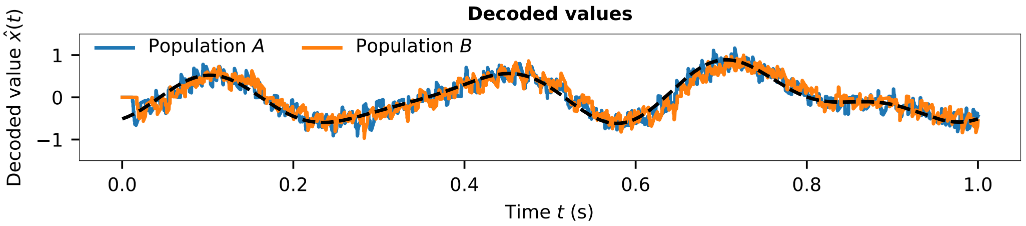


Communication Channel Experiment: Populations in series

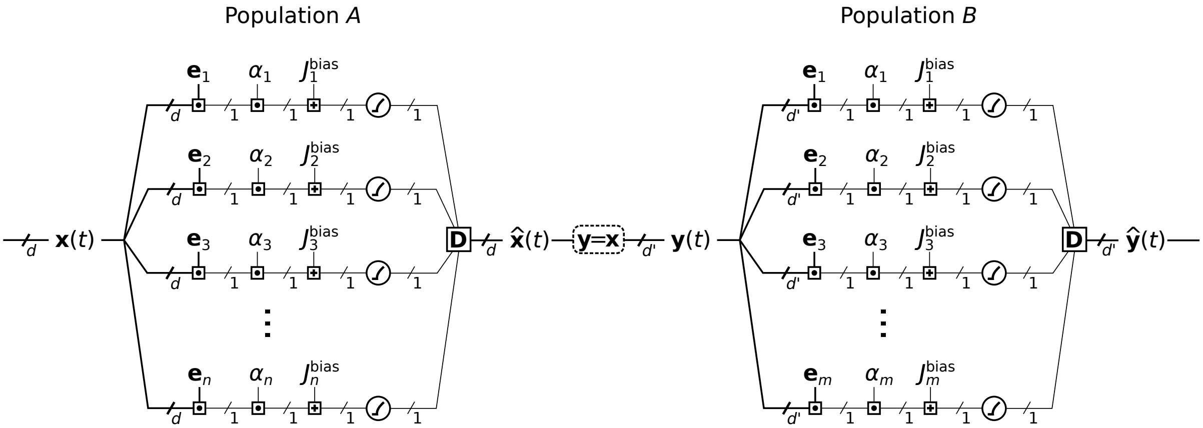


Communication Channel Experiment: Populations in series

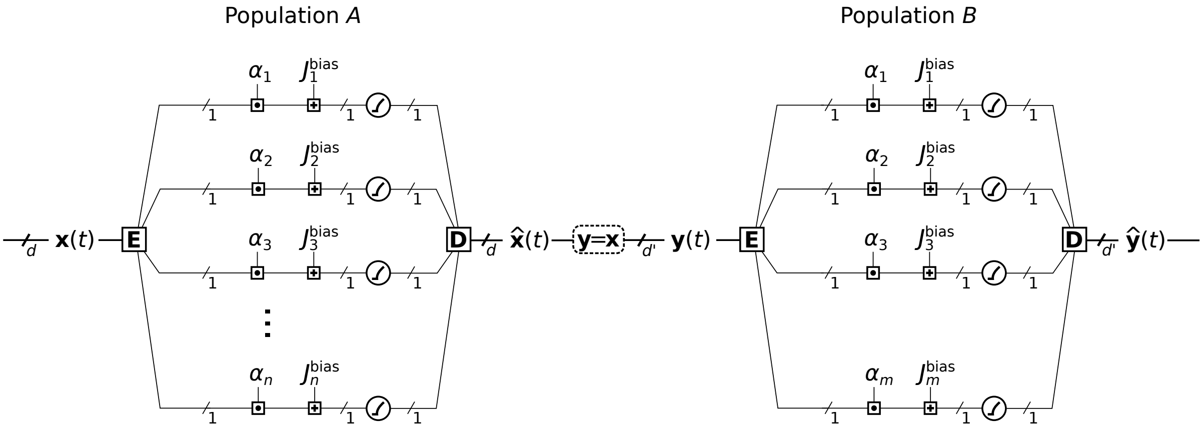




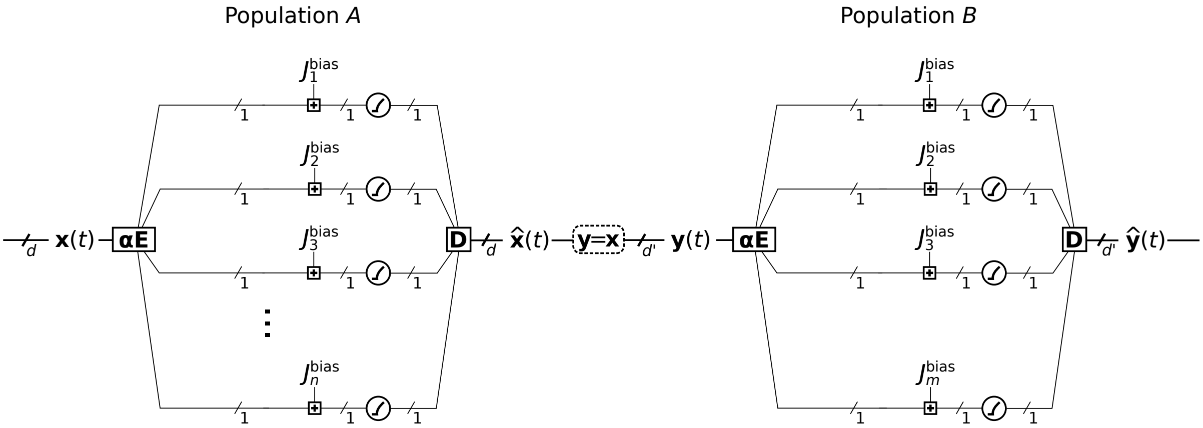
Computing Synaptic Weights: Step 1 – Encoding Matrix



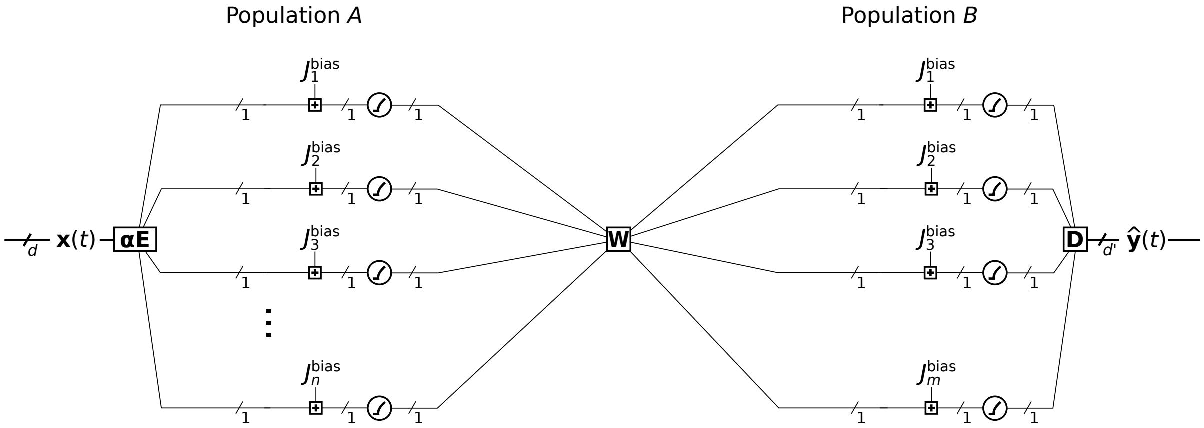
Computing Synaptic Weights: Step 1 – Encoding Matrix



Computing Synaptic Weights: Step 2 – Scaled Encoding Matrix



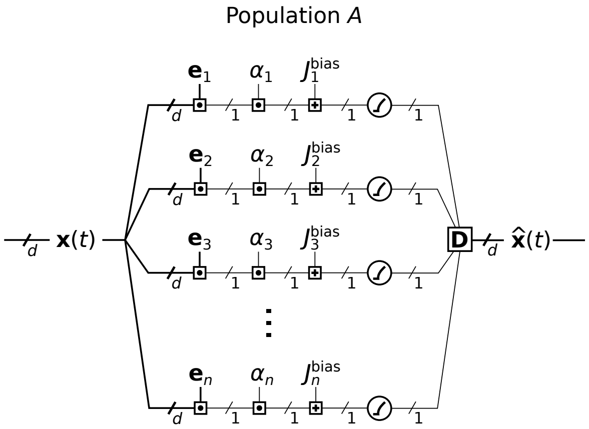
Computing Synaptic Weights: Step 3 – $\mathbf{W} = \mathbf{E}\mathbf{D}$

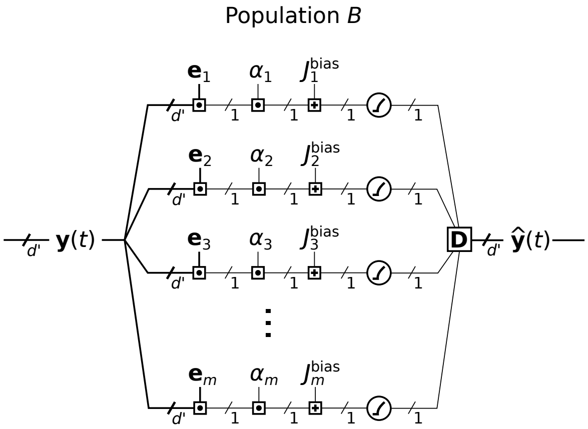


Computational Complexity

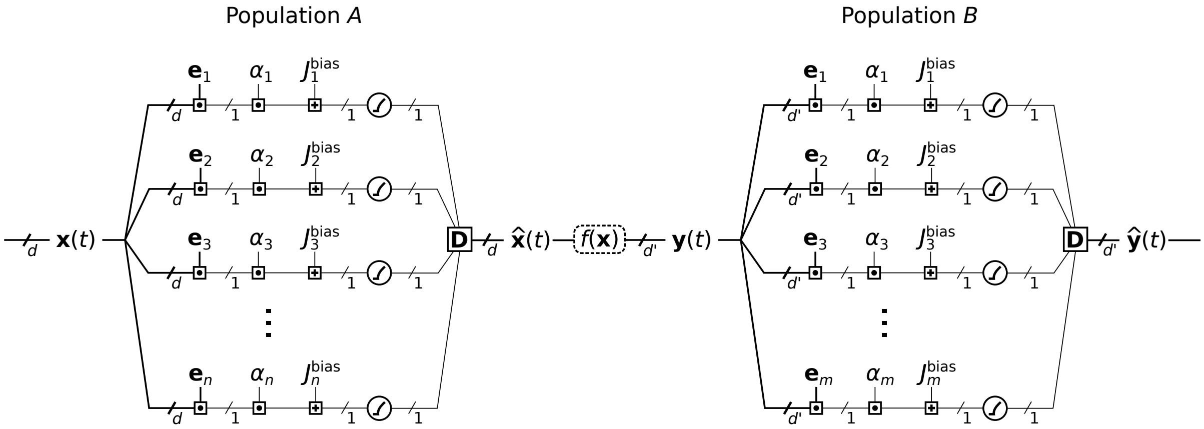
- Weights - multiplying $\mathbf{a} \in \mathbb{R}^n$ with $\mathbf{W} \in \mathbb{R}^{m \times n}$ is $\mathcal{O}(nm)$ i.e., $\approx \mathcal{O}(n^2)$
- Decoding - $\hat{\mathbf{x}} = \mathbf{D}\mathbf{a}$ is $\mathcal{O}(dn)$
- Encoding - $\mathbf{J} = \mathbf{E}\hat{\mathbf{x}} + \mathbf{J}_{\text{bias}}$ is $\mathcal{O}(dm)$
- Encoding/Decoding - $\mathcal{O}(d(n+m))$ or $\approx \mathcal{O}(dn)$ for $n = m$
- So if d is small we get a linear complexity $\mathcal{O}(n)$
- Therefore, sequential decoding and re-encoding saves a lot of time compared to using actual synaptic weights
- One reason why Nengo is so fast compared to other SNN simulators

Computing Functions

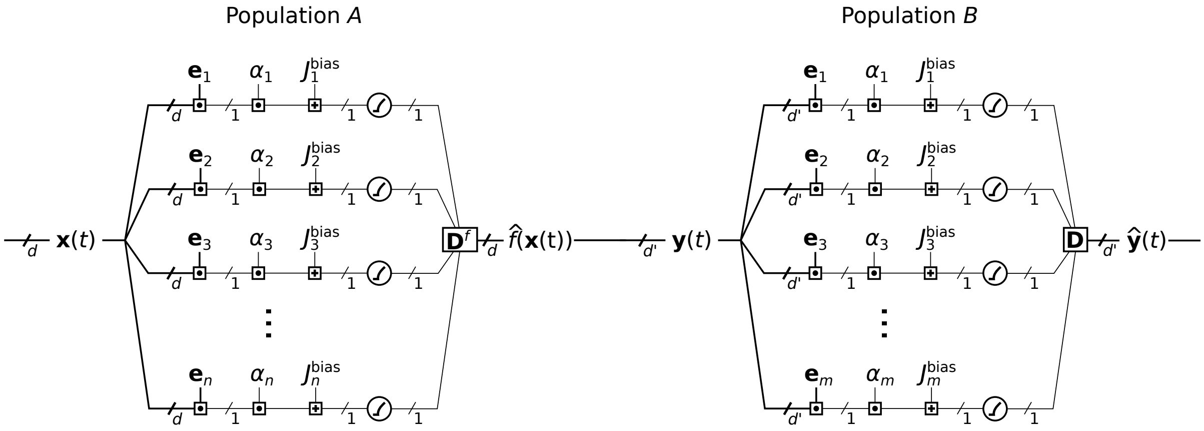

Population A
$\mathbf{e}_1$ α_1 J_1^{bias}
$\mathbf{e}_2$ α_2 J_2^{bias}
$\mathbf{e}_3$ α_3 J_3^{bias}
$\mathbf{e}_n$ α_n J_n^{bias}
d
1
$\mathbf{x}(t)$
D
$\hat{\mathbf{x}}(t)$


Population B
$\mathbf{e}_1$ α_1 J_1^{bias}
$\mathbf{e}_2$ α_2 J_2^{bias}
$\mathbf{e}_3$ α_3 J_3^{bias}
$\mathbf{e}_m$ α_m J_m^{bias}
d'
1
$\mathbf{y}(t)$
D
$\hat{\mathbf{y}}(t)$

Computing Functions


Population A
Population B
$\mathbf{x}(t)$
$\hat{\mathbf{x}}(t)$
$f(\mathbf{x})$
$\mathbf{y}(t)$
$\hat{\mathbf{y}}(t)$
$\mathbf{e}_1$ α_1 J_1^{bias}
$\mathbf{e}_2$ α_2 J_2^{bias}
$\mathbf{e}_3$ α_3 J_3^{bias}
$\mathbf{e}_n$ α_n J_n^{bias}
$\mathbf{e}_m$ α_m J_m^{bias}
D
d
d'
1

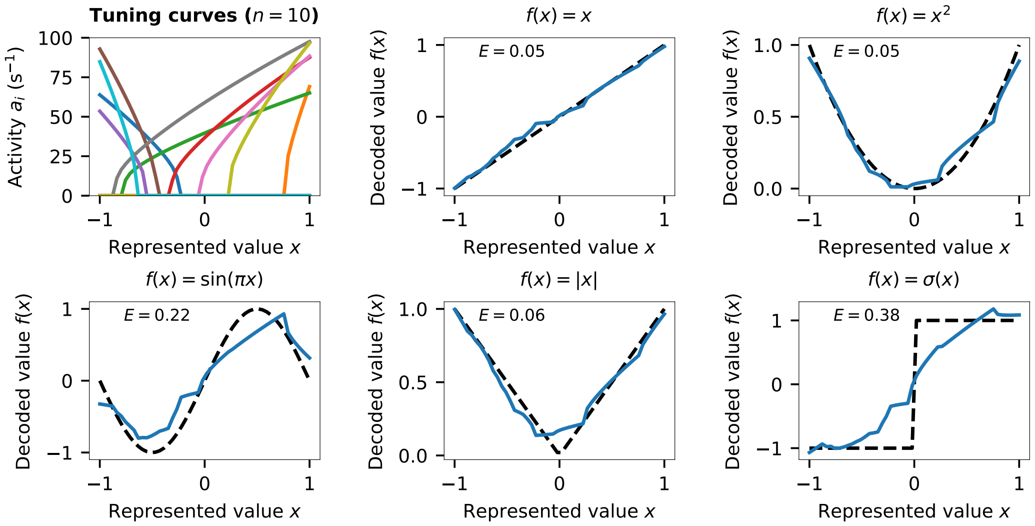
Computing Functions



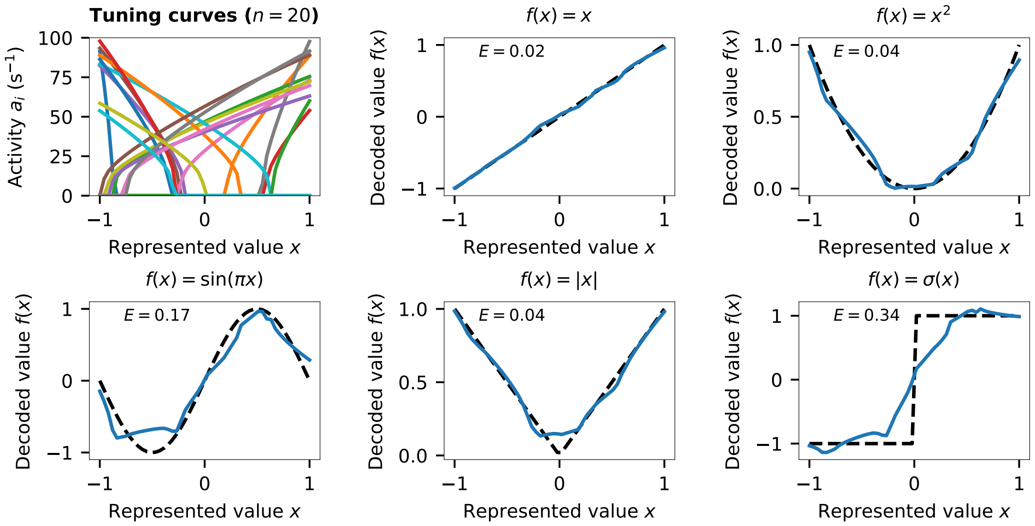
Population A
Population B
$\mathbf{e}_1$ α_1 J_1^{bias}
$\mathbf{e}_2$ α_2 J_2^{bias}
$\mathbf{e}_3$ α_3 J_3^{bias}
$\mathbf{e}_n$ α_n J_n^{bias}
$\mathbf{e}_m$ α_m J_m^{bias}
d
d'
1
$\mathbf{x}(t)$
$\mathbf{D}^f$
$\hat{f}(\mathbf{x}(t))$
$\mathbf{y}(t)$
$\mathbf{D}$
$\hat{\mathbf{y}}(t)$

Function Decoder $\mathbf{D}^f = \left((\mathbf{A}\mathbf{A}^\mathsf{T} + N\sigma^2\mathbf{I})^{-1}\mathbf{A}\mathbf{Y}^\mathsf{T}\right)^\mathsf{T}$, where $\left(\mathbf{Y}\right)_{ik} = \left(f(\mathbf{x}_k)\right)_i$

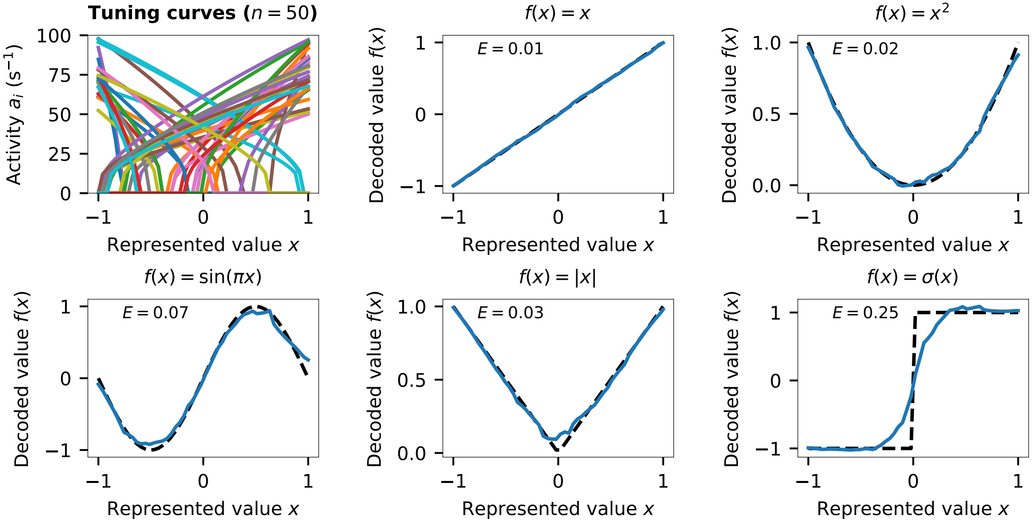
Decoding Functions – Using a Few Neurons



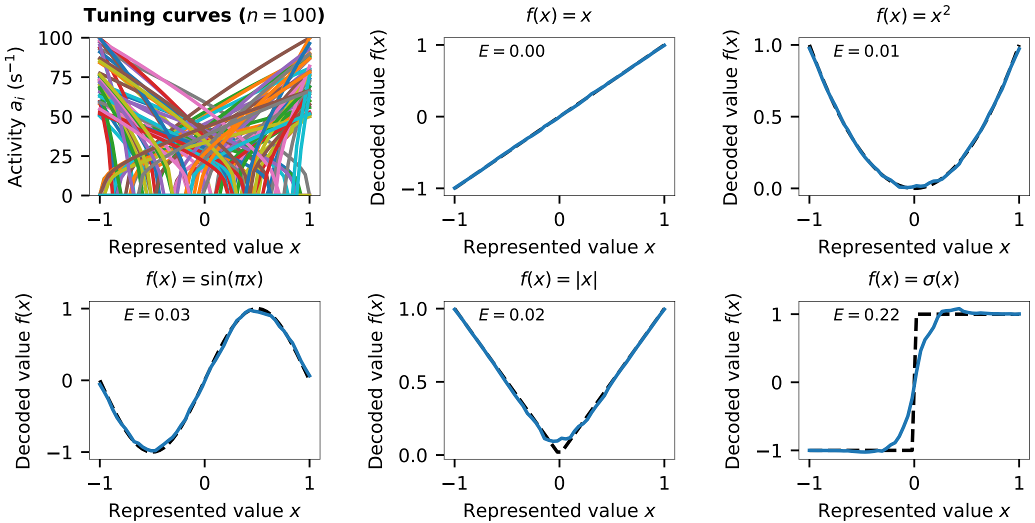
Decoding Functions – Using More Neurons



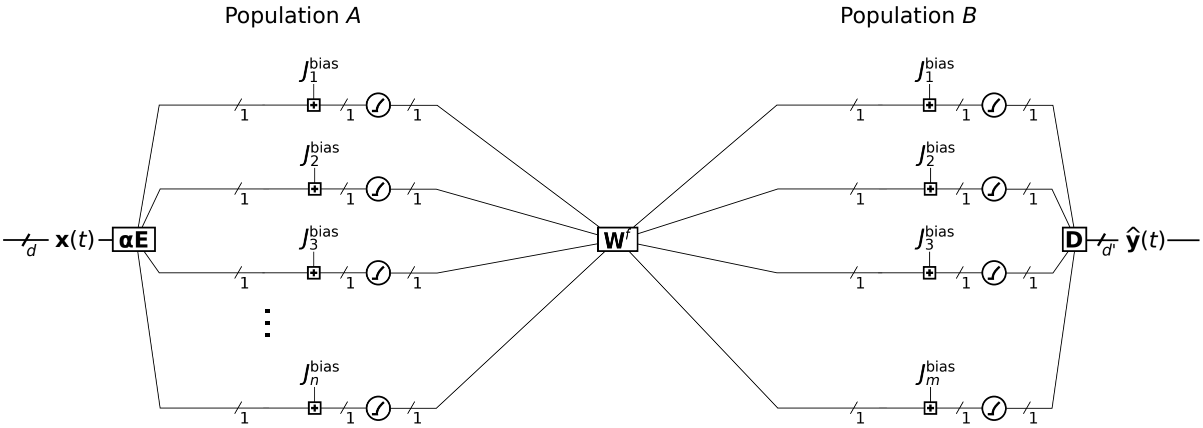
Decoding Functions – Using More Neurons



Decoding Functions – Using More Neurons



Computing Functions – Weight Matrix



$$\mathbf{W}^f = \mathbf{E}\mathbf{D}^f$$

Recipe for any feedforward transformation

1. Define encoding for two populations (input/output)
2. Write the transformation with the represented input variables
3. Solve for the $\mathbf{D}^f$ for that transformation for the input population
4. Sub 3. into the encoding for the output variable

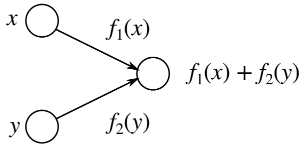
Computing Multivariate Functions

○ Homogenous population ◌ Heterogenous population

→ Linear connection ⊣ Inh. connection —● Exc. connection

Linear Superposition

$$W^{f_1}\mathbf{a}_1(\mathbf{x}) + W^{f_2}\mathbf{a}_2(\mathbf{y})$$



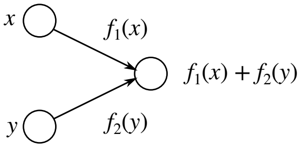
Computing Multivariate Functions

○ Homogenous population ◌ Heterogenous population

→ Linear connection ⊣ Inh. connection —● Exc. connection

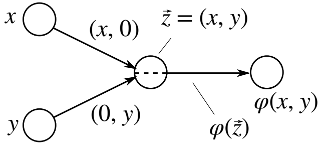
Linear Superposition

$$W^{f_1}\mathbf{a}_1(\mathbf{x}) + W^{f_2}\mathbf{a}_2(\mathbf{y})$$



Nonlinear Functions

Multi-dimensional $\mathbf{z}$



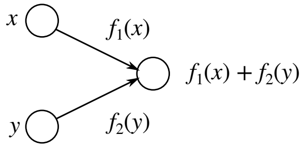
Computing Multivariate Functions

○ Homogenous population ◎ Heterogenous population

→ Linear connection ⊣ Inh. connection ● Exc. connection

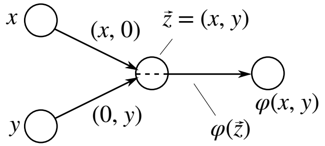
Linear Superposition

$W^{f_1}\mathbf{a}_1(\mathbf{x}) + W^{f_2}\mathbf{a}_2(\mathbf{y})$



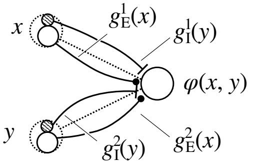
Nonlinear Functions

Multi-dimensional $\mathbf{z}$



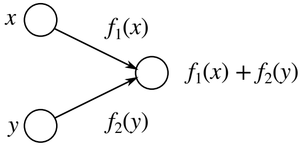
(Dendritic Computation)

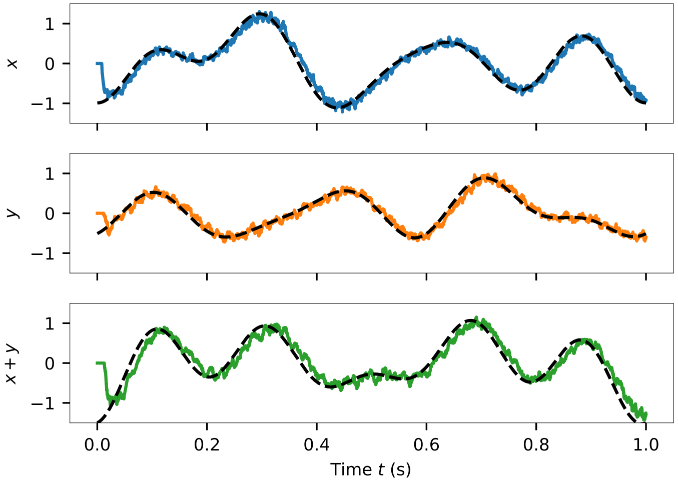
Exploit dendritic nonlinearity



Computing Multivariate Functions – Linear Superposition

Linear Superposition

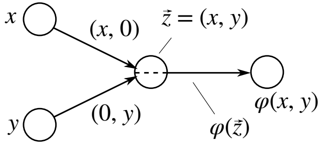


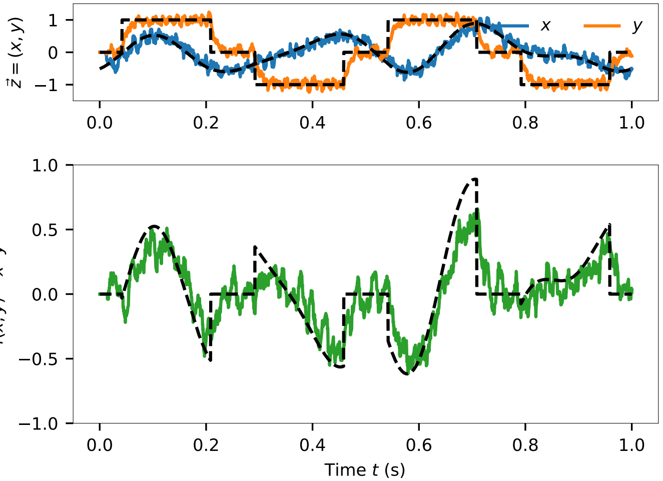


Computing Multivariate Functions – Multiplication

Nonlinear Functions

Multi-dimensional $\mathbf{z}$

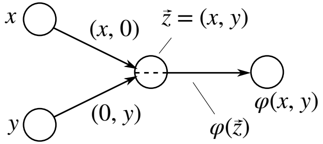




Computing Multivariate Functions – Multiplication

Nonlinear Functions

Multi-dimensional $\mathbf{z}$



Multiplication is useful...

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

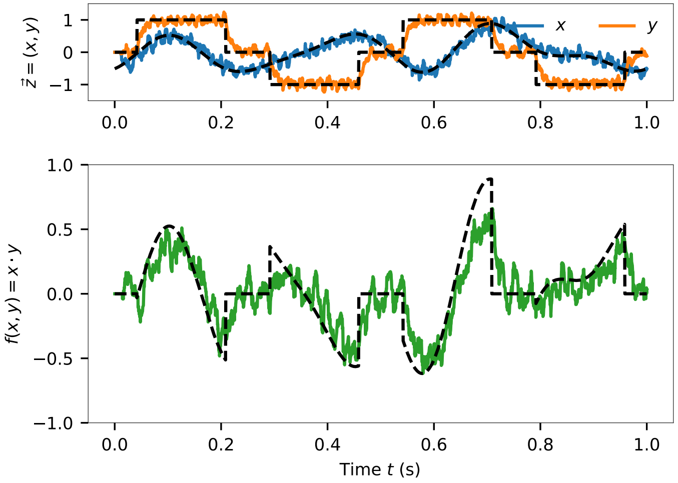


Image sources

Title slide
“Yellow Butterfly”
Author: Albert Bierstadt, circa 1890.
From Wikimedia.