COMP 400 Final Report: RatSlam 2.0

Ezra Huang* Jonathan Lamontagne-Kratz* Olivier Lefevre*
School of Computer Science
McGill University
{ezra.huang,olivier.lefevre,jonathan.lamontagne-kratz}@mail.mcgill.ca

Isabeau Premont-Schwarz
School of Computer Science
McGill University
isabeau.premont-schwarz@mcgill.ca

* These authors contributed equally. Ordering was determined alphabetically.

May 5, 2025

SLAM Approaches: Traditional vs. Bio-inspired

- **Traditional Methods** [6]:
  - Probabilistic frameworks: Kalman filters, particle filters
  - GraphSLAM optimization
- **Brain-inspired Success**:
  - RatSLAM [2]:
    - Hippocampal model integration
    - Visual + odometric fusion
    - Real-time mapping [3]

VectorHaSH Foundations

Core Architecture [1]

- Neocortical-entorhinal-hippocampal network
- Dual cellular scaffold:
  - Grid cell spatial encoding
  - Hippocampal memory formation
- Pseudoinverse learning mechanism

Key Innovation

- High-capacity memory storage
- Biologically plausible associatave learning

Ezra

Key Contributions & Roadmap

- **3D Scaffold Generalization**
  - Extend VectorHaSH to 3D/non-square modules
  - Preserve capacity properties: large basins, fixed points
  - Hippocampal cells scale with module count, not size
- **Initialization Analysis**
  - Compare methods from [1] and [4]
  - Demonstrate equivalent hippocampal cell requirements
- **Continuous State Integration**
  - Adapt RatSLAM's velocity shift mechanism
  - Enable smooth transitions between grid states
- **Associative Learning**
  - Enhanced Hebbian learning variants
  - Biologically-plausible pseudoinverse algorithm
- **Integrated System Validation**
  - Full evaluation in AnimalAI [7]

Ezra

Extended Grid-Hippocampal Scaffold

Extended Grid-Hippocampal Scaffold

Core Extensions from VectorHaSH [1]

- **3D Pose Encoding**:
  - Grid states represent (x, y, θ) pose
  - Non-square modules: $\lambda_x \neq \lambda_y \neq \lambda_\theta$
- **Module Specifications**:
  - M modules with sizes $\boldsymbol{\lambda}^{(m)} = (\lambda_x, \lambda_y, \lambda_\theta)^{(m)}$
  - Pairwise coprime dimensions: $\forall d \in \{x, y, \theta\}, \lambda_d^{(i)} \perp \lambda_d^{(j)} (i \neq j)$

Ezra

Extended Grid-Hippocampal Scaffold

State Formulation

- Module state: $s^{(m)} \in [0,1]^{\lambda_x^{(m)} \times \lambda_y^{(m)} \times \lambda_\theta^{(m)}} \qquad m \in \{1, \ldots, M\}$
- Flattened encoding: $g^{(m)} \in [0,1]^{\prod \lambda_i^{(m)}}$
- Combined state vector: $g := [g^{(1)} \| \cdots \| g^{(M)}]$
- Vector length: $N_g = \sum_{m=1}^{M} \prod_i \lambda_i^{(m)}$
- Fixed points: $N_{patts} = \prod_{m=1}^{M} \prod_i \lambda_i^{(m)}$

Ezra

Continuous Grid States

Continuous Grid States

VectorHaSH Approach [1]

- Gaussian activity blobs
- Neural sheet representation

Our Method (RatSLAM Inspired) [2]

- Competitive attractor network
- Smoothing function application

- State space: $\Omega^{(m)} = \{(x, y, \theta) \in \{0, \ldots, \lambda_x^{(m)} - 1\} \times \{0, \ldots, \lambda_y^{(m)} - 1\} \times \{0, \ldots, \lambda_\theta^{(m)} - 1\}\}$
- Interpret as categorical distribution

Ezra

Continuous Grid States

Marginal Distribution Calculation

For $k \in \{0, \ldots, \prod \lambda_x^{(m)} - 1\}$:

$$p(x = k) = \prod_{m=1}^{M} p^{(m)}(x = k \bmod m)$$

- Analogous computation for $p(y)$, $p(\theta)$
- Maintains full pose uncertainty estimates

Ezra

Extended Initialization Methods

Grid-Hippocampal Weight Initialization

Goals

- Both methods achieve:
  - Desired hippocampal sparsity (ψ)
  - Preservation of scaffold properties

Compared Approaches

- **VectorHaSH Method** [1]:
  - $W_{gh} \sim \mathcal{N}(0, \sigma^2)$ with sparsity control
  - Standard normal base distribution
- **Alternative Method** [4]:
  - Non-zero mean initialization: $\mu \neq 0$
  - Achieve desired sparsity after a ReLU

Olivier

Grid-Hippocampal Velocity Shifts

Velocity Shift Mechanism Comparison

VectorHaSH Approach [1]

- Simple activity translation
- Discrete state transitions

Our RatSLAM-inspired Method [2]

- Continuous odometry integration
- Activity spreading mechanism
- Handles small incremental movements

Jonathan

Ratshift Operational Pipeline

Velocity Scaling

$$\begin{cases} v_x = v\cos(\theta)k_x \\ v_y = v\sin(\theta)k_y \\ v_\theta = \omega k_\theta \end{cases}$$

Component Decomposition

For each dimension $d \in \{x, y, \theta\}$: $\begin{cases} \delta_x, \delta_{fx} = \lfloor v_x \rfloor, v_x - \lfloor v_x \rfloor \\ \delta_y, \delta_{fy} = \lfloor v_y \rfloor \; v_y - \lfloor v_y \rfloor, \\ \delta_\theta, \delta_{f\theta} = \lfloor v_\theta \rfloor, v_\theta - \lfloor v_\theta \rfloor \end{cases}$

Jonathan

Activity Update Formulation

Alpha Weight Calculation

$$\alpha_{ijk} = g(\delta_{fx}, i) \cdot g(\delta_{fy}, j) \cdot g(\delta_{f\theta}, k)$$

Where:

$$g(a, b) = \begin{cases} 1 - a & b = 0 \\ a & \text{otherwise} \end{cases}$$

State Transition Equation

$$s_{nlm} = \sum_{i,j,k=0}^{1} \alpha_{ijk} \cdot s_{i'j'k'}$$

$$\begin{cases} i' = (n + i - \delta_x) \mod \lambda_x \\ j' = (l + j - \delta_y) \mod \lambda_y \\ k' = (m + k - \delta_\theta) \mod \lambda_\theta \end{cases}$$

Smoothing Methods for SLAM

Jonathan

Smoothing Methods for SLAM

Key Requirements

- Maintain probability distribution properties
- Prevent uniform distribution convergence during:
  - No movement periods
  - Small odometry inputs

Design Challenge

- Balance information preservation
- Prevent probability mass dispersion

Jonathan

Smoothing Methods for SLAM

Argmax Smoothing [1]

$$i, j, k = \arg\max_{ijk}(s), \quad s'_{x,y,\theta} = \delta_{ix}\delta_{jy}\delta_{k\theta}$$

- Kronecker delta implementation
- Drawback: Information loss in small movements

Polynomial Smoothing

$$\phi(x)_{ijk} = \frac{x_{ijk}^{\kappa}}{\sum x_{xy\theta}^{\kappa}}$$

- Hyperparameter κ controls sharpness
- Benefit: Preserves zero-mass cells

Jonathan

Probabilistic Smoothing Method

Softmax Smoothing

$$\sigma(x)_{ijk} = \frac{e^{x_{ijk}/\tau}}{\sum_{xy\theta} e^{x_{ijk}/\tau}} \quad s' = \sigma(s)$$

- Temperature τ controls distribution sharpness
- Neural network-inspired approach

Jonathan

RatSLAM Smoothing Process [2]

1. **Spatial Update**

$$s_{xy\theta} += \sum_{i,j=-r}^{r} \epsilon_{ij} \cdot s_{x+i,y+j,\theta}$$

Gaussian kernel ϵ with radius $r = \lfloor \frac{\lambda_x}{2} \rfloor$

2. **Orientation Update**

$$s_{xy\theta} += \sum_{k=-\gamma}^{\gamma} \delta_k \cdot s_{x,y,\theta+k}$$

1D Gaussian δ over γ-neighborhood

Jonathan

RatSLAM Smoothing (Cont.)

3. **Global Inhibition**

$$s_{xy\theta} = \max\left(0, s_{xy\theta} + \lambda(s_{xy\theta} - \max s_{x'y'\theta'})\right)$$

4. **Normalization**

$$s_{xy\theta} = \frac{s_{xy\theta}}{\sum s_{x'y'\theta'}}$$

Implementation Notes

- Circular padding for spatial updates
- Competitive dynamics maintain activation peaks

Jonathan

Hippocampal-Sensory Learning

Hippocampal-Sensory Learning

Objective

Continuously update W_{hs} and W_{sh} to minimize:

$$\mathcal{L}(S, W_{sh}H) \quad \text{and} \quad \mathcal{L}(H, W_{hs}S)$$

- Notation: s^i = sensory input, h^i = hippocampal state
- Sets: H = all hippocampal states, S = all sensory inputs

Olivier

Hebbian Learning Variants

Basic Hebbian

Update rules for new pair (s^j, h^j):

$$W_{sh} \leftarrow W_{sh} + \frac{s^j (h^j)^T}{\|h^j\|}$$

$$W_{hs} \leftarrow W_{hs} + \frac{h^j (s^j)^T}{\|s^j\|}$$

Olivier

Zero-Centered Hebbian

Interference Problem

For recovered sensory input s^*:

$$s_i^* = s_i^\nu + \sum_{\mu \neq \nu} \sum_j \frac{s_i^\mu h_j^\mu h_j^\nu}{\|h^\mu\|^2}$$

- Non-zero $\bar{h}$ causes residual correlations
- Rectification ($h_j \geq 0$) breaks orthogonality assumption
- Must compute $\bar{h} = E[h]$
- Center hippocampal states: $h \leftarrow h - \bar{h}$
- Ensures $E[h_j - \bar{h}_j] = 0$

Ezra

Zero-Centered Hebbian (Cont.)

Modified Learning Rules

$$W_{sh} \leftarrow W_{sh} + \frac{s^j(h^j - \bar{h})^T}{\|h^j\|}$$

$$W_{hs} \leftarrow W_{hs} + \frac{(h^j - \bar{h})(s^j)^T}{\|s^j\|}$$

- Recall equations become:

$$h - \bar{h} = W_{hs}s \quad \text{and} \quad s = W_{sh}(h - \bar{h})$$

Ezra

Pseudoinverse Methods

Analytic (VectorHaSH [1])

$$W_{hs} = HS^{+}, \quad W_{sh} = SH^{+}$$

- **Must store past sensory data**
- S is 0 matrix with non-zero entries for sensory inputs seen
- Fixed hippocampal matrix H

Olivier

Iterative Pseudoinverse Algorithm [5]

Bidirectional Iterative Pseudoinverse Learning

Input: Sensory inputs S, Hippocampal states H
Output: Updated weights W_{hs}, W_{sh}
foreach *new pair* (s^j, h^j) **do**
    Update W_{hs}:

$$b_k = \frac{\Theta_{hs} \cdot s^j}{1 + (s^j)^T \Theta_{hs} s^j}$$

$$\Theta_{hs} \leftarrow (\Theta_{hs} - \Theta_{hs} s^j b_k^T$$

$$W_{hs} \leftarrow W_{hs} + (h^j - W_{hs} s^j) b_k^T$$

    Update W_{sh} similar procedure switch h's and s's
end

Jonathan

RatSLAM 2.0 Algorithm Variations

Memory Storage Strategies

Hard Store

- Apply argmax smoothing
- Use smoothed state for learning:

$$g_{store} = \arg\max(g)$$

Soft Store

- Use current grid state directly:

$$g_{store} = g$$

Olivier

State Merging Methods

Additive Merging

$$g = g_s + g_o$$

where g_s and g_o are concatenated and flattened states across the modules

Multiplicative Merging

$$g = g_s \odot g_o \quad \text{(element-wise product)}$$

Olivier

When to store

Always

Here we store the sensory inputs to corresponding states at all time steps.

When new

When the mean μ along each dimension by $\mu = E[X] = Arg(m_1)$. Where m_1 is the 1st moment of a circular distribution (expected value).

Olivier

RatSLAM 2.0 Core Algorithm

$s \leftarrow$ InitializeModules(λ)
$g \leftarrow$ FlattenGrid
$W_{hg} \leftarrow$ InitializeWeights(μ, σ, γ)
$W_{gh} \leftarrow$ Calculate $W_g h$

while *true* **do**
    take action u_t
    $dx, dy, d\theta \leftarrow$ Get Odometry()
    $g' :=$Ratshift($g, dx, dy, d\theta$)
    $s \leftarrow$ Get Sensory()
    if Esimate certainty(g', k) $\leq p_{threshold}$ **then**
        $\bar{h} \leftarrow$hippocampal from sensory(s)
        $\bar{g} \leftarrow$grid from hippocampal(h)
        $g \leftarrow$Additive Merge((g'), *curr*) or Multiplicative Merge((g'), *curr*)
    else
        Store Memory(g, s)
        $g \leftarrow$Additive Merge(g', g) or Multiplicative Merge(g', g)
    end
end

Olivier

Certainty

Calculating Certainty

- Use the p-th moment of a circular distribution
- For each grid module, we can calculate the mean μ along each dimension by $\mu = E[X] = Arg(m_1)$.

$$m_p = \frac{1}{n}\sum_{k=1}^{n}(e^{ix_k})^p = \frac{1}{n}\sum_{k=1}^{n}\cos(px_k) + i\sin(px_k) \tag{1}$$

$$P(|X-\mu| > k) \leq \sum_{i \notin \{\lceil \mu - k \rceil, \ldots, \lfloor \mu + k \rfloor\}} P(X = i) \leq p \tag{2}$$

Olivier

Testing of Grid-Hippocampal Initialization Methods

Testing of Grid-Hippocampal Initialization Methods

Experimental Validation

- Compared VectorHaSH normal vs. non-zero mean initialization
- Measured hippocampal neuron activation error through:
  - Grid $\rightarrow$ Hippocampal projection
  - Denoising process
  - Hippocampal $\rightarrow$ Grid back-projection
- Tested with varied parameters:
  - Hippocampal cell count (50-1000)
  - Pattern subsets (up to theoretical capacity)

Ezra

Testing of Grid-Hippocampal Initialization Methods

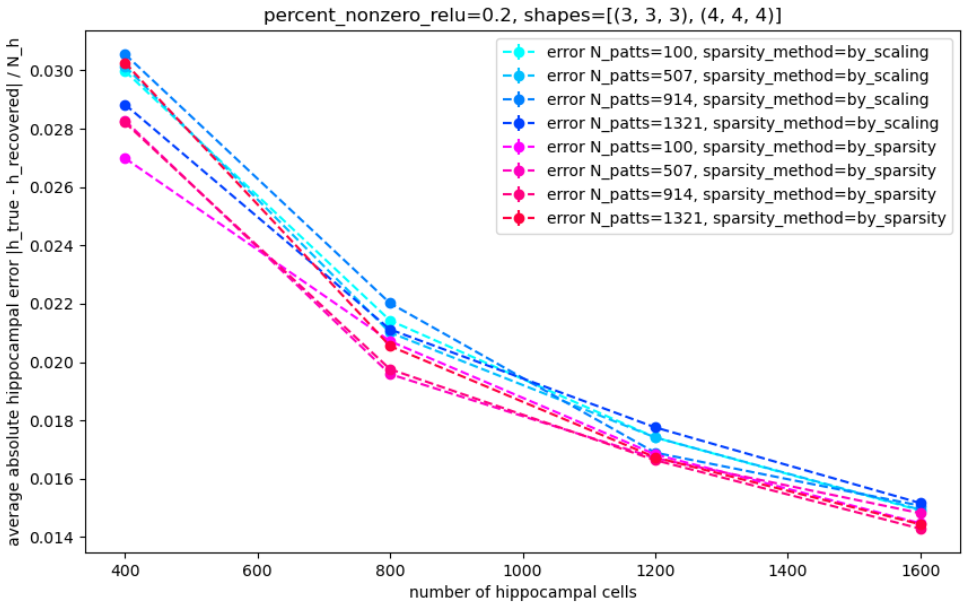

Figure: Equivalent performance between initialization methods

Ezra

Testing of Grid-Hippocampal Initialization Methods

Critical Findings

- Both methods preserve VectorHaSH's exponential error scaling:

$$\text{Error} \propto \exp(-N_h)$$

- Marginal differences (± 0.001) between methods
- Capacity-independent performance below 1728 patterns

Ezra

Properties of Extended Grid-Hippocampal Scaffold

Dimensional Independence

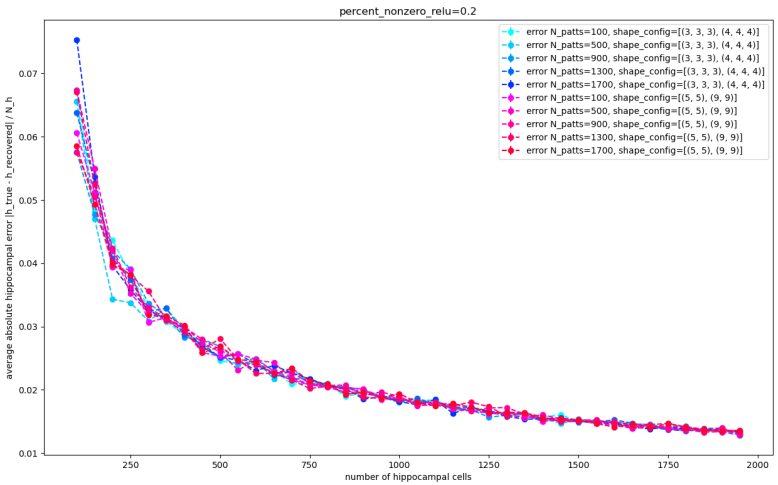

Figure: Equivalent error rates for 2D/3D modules

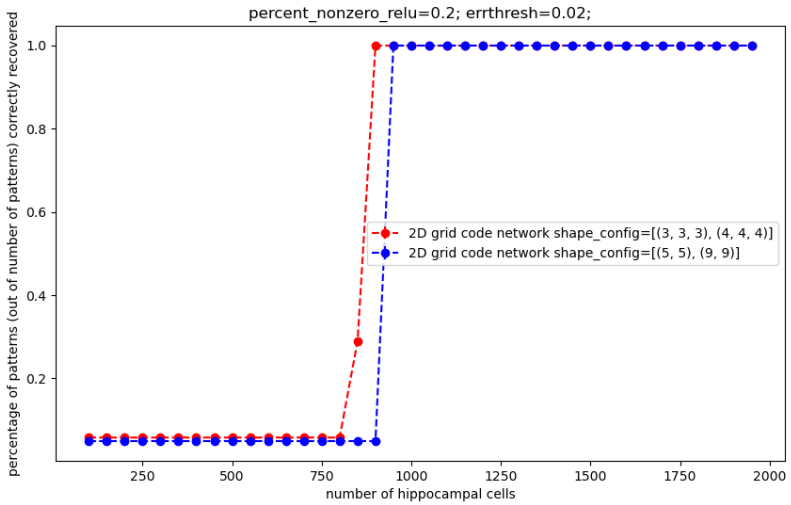

Figure: Consistent capacity requirements

Ezra

Modular Scaling Property

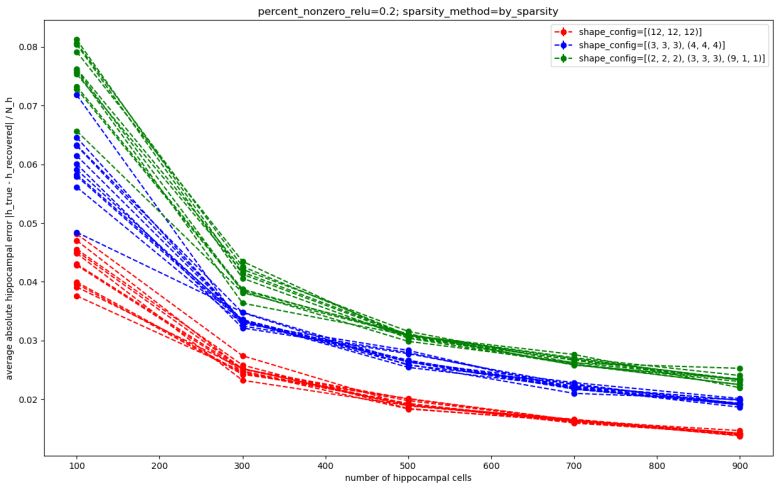

Figure: Error vs. module count

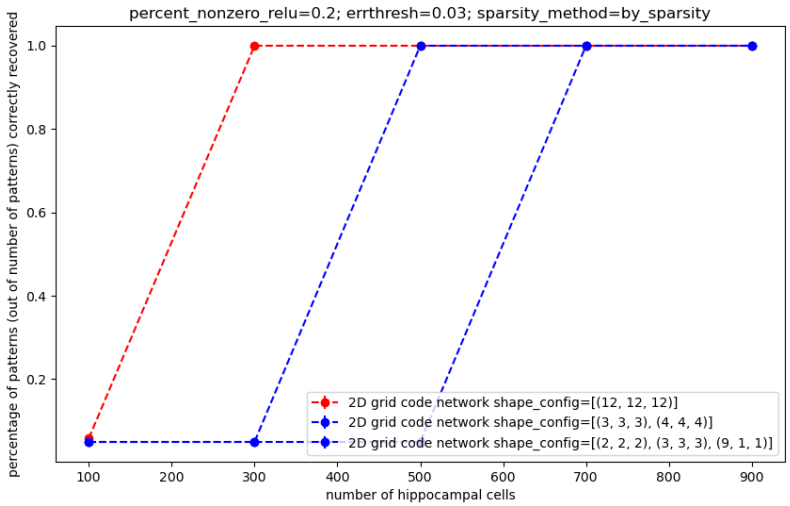

Figure: Capacity vs. modules

Ezra

Preserved VectorHaSH Properties

Core Maintained Characteristics

- **Dimensional Neutrality**:
  - 2D/3D modules show identical performance
  - Error independent of spatial encoding complexity
- **Modular Scaling**:
  - Linear relationship: cells $\propto$ modules

Ezra

Iterative vs. Analytic Pseudoinverse Performance

Iterative vs. Analytic Pseudoinverse Performance

Experimental Setup

- Benchmark against MNIST pattern recovery
- Metric: average L2 error per neuron
- Tested beyond theoretical capacity limits

Jonathan

Iterative vs. Analytic Pseudoinverse Performance

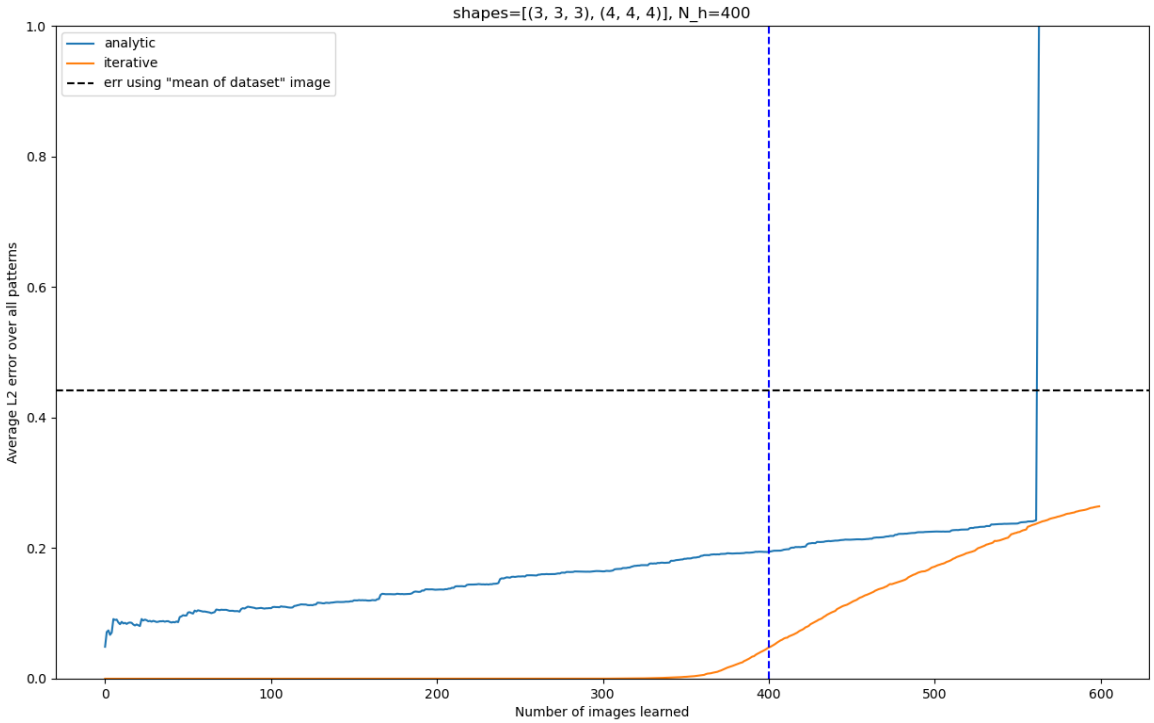

Figure: Superior performance of iterative method

Jonathan

Iterative vs. Analytic Pseudoinverse Performance

Iterative Method Advantages

- Sustained performance beyond 560 patterns
- Iterative method enables graceful degradation
- More suitable for real-time SLAM applications

Analytic Method Limitations

- **Have to store all previously seen images**
- Early catastrophic failure at $\sim$560 patterns
- Numerical instability in PyTorch's pinv()

Jonathan

Testing of Hebbian Learning Variations

Testing of Hebbian Learning Variations

MNIST Testing

- Whitened MNIST dataset
- Sequential image learning with metrics:
  - Average L2 over Dataset
  - L1: First image error
  - L1: Last image error
- 10-run average with $\pm 1\sigma$ bands
- Baseline: Dataset mean error

Ezra

Testing of Hebbian Learning Variations

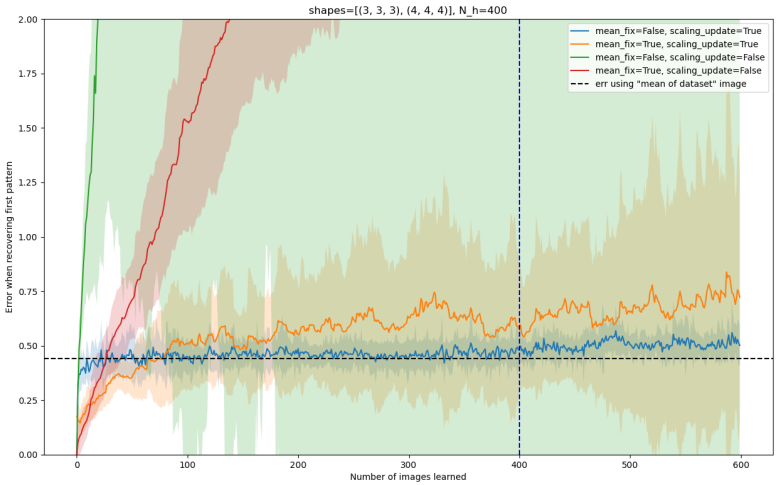

Figure: First image recall degradation

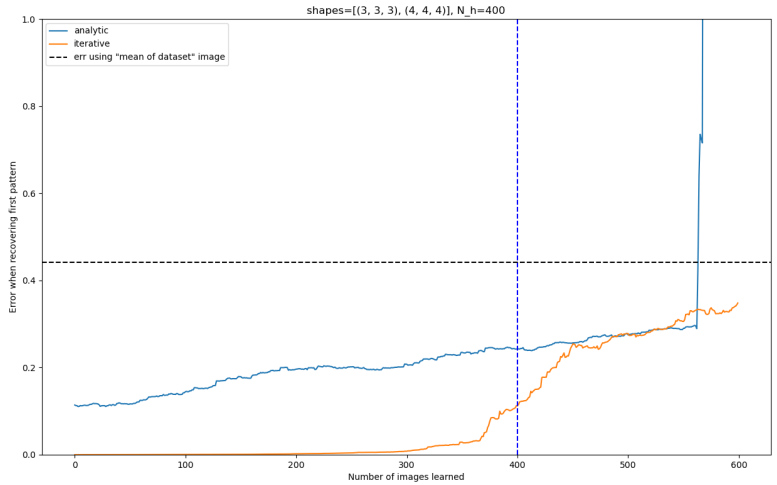

Figure: Pseudoinverse stability

Key Finding

Hebbian methods fail to outperform mean baseline after 100 patterns

Ezra

Testing of Hebbian Learning Variations

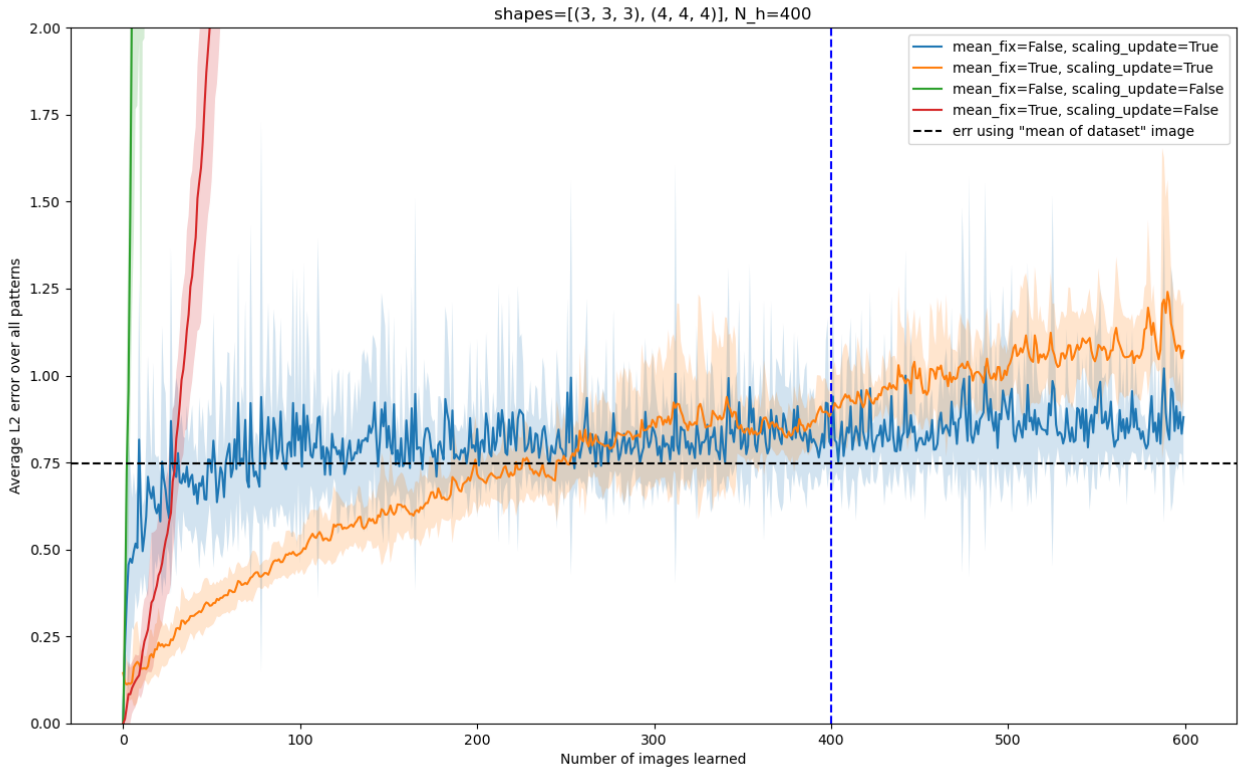

Figure: Dataset error with/without mean correction

Ezra

Testing of Hebbian Learning Variations

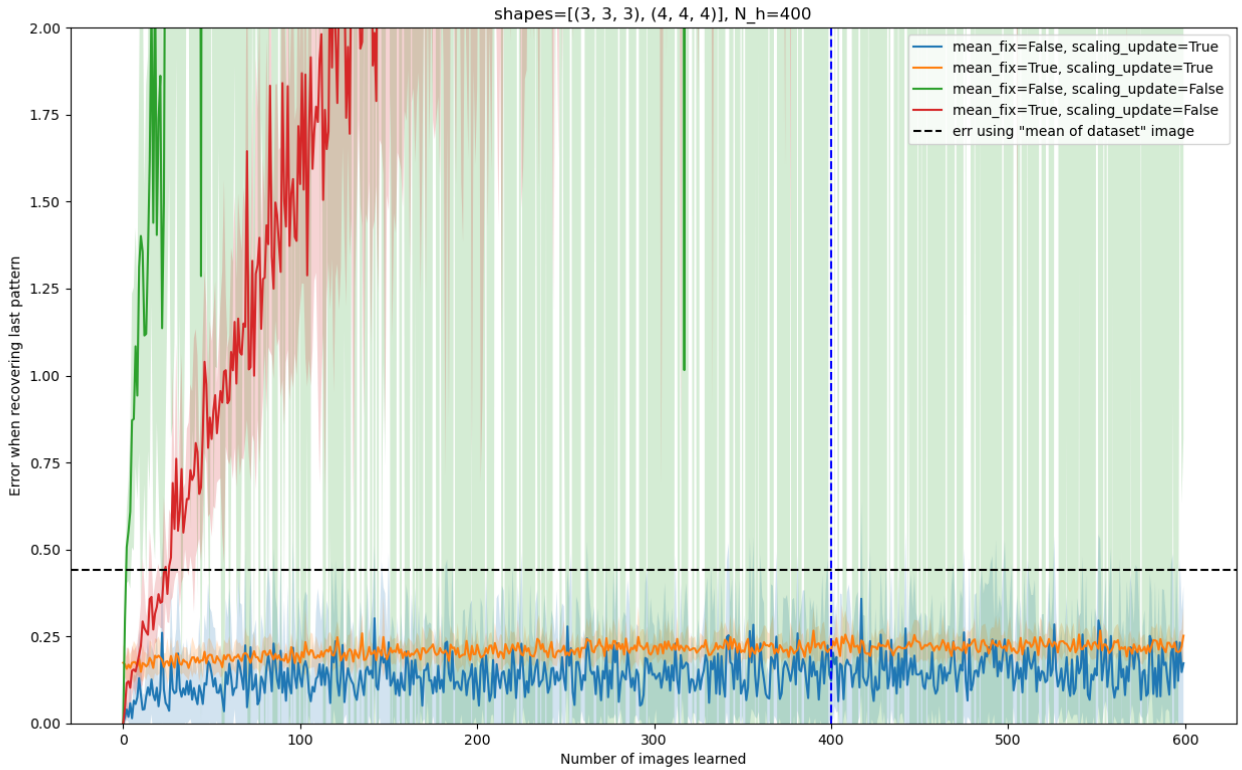

Figure: Last image error without reinforcement

Ezra

RatSLAM 2.0 Algorithm Testing

Olivier

RatSLAM 2.0 Algorithm Testing

Kidnapping Scenario Test

- 10 steps forward $\rightarrow$ 90° turn $\rightarrow$ reset without odometry
- Recovery validation phase: 10 additional steps
- Metrics: Pose error (x, y, θ)

Tested Variations

- Hard vs. soft storing
- Multiplicative vs. additive merging
- Smoothing methods:
  - Softmax (τ-controlled)
  - Polynomial (κ-parameterized)
  - RatSLAM CAN dynamics

Olivier

RatSLAM 2.0 Algorithm Testing

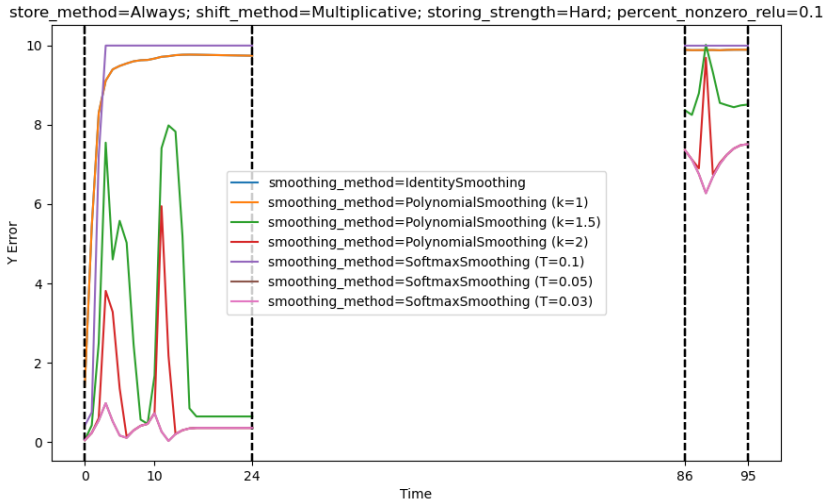

Figure: y dimension error with various smoothing methods after Ratshift

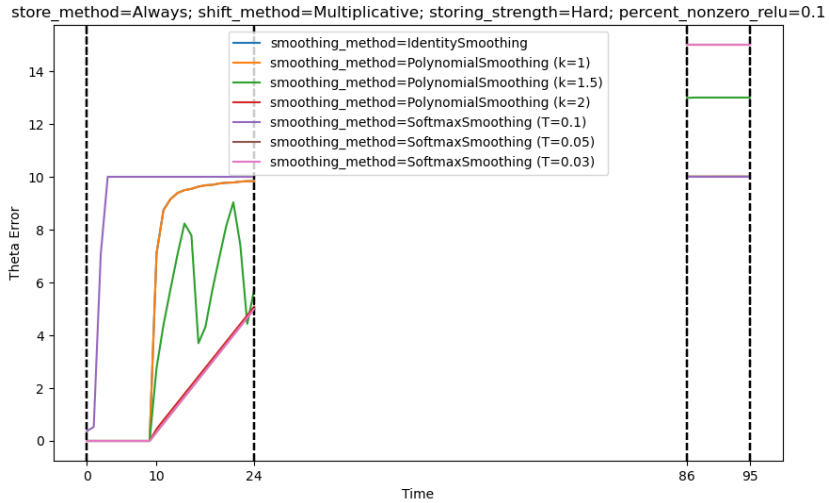

Figure: θ dimension error with various smoothing methods after Ratshift

Olivier

RatSLAM 2.0 Algorithm Testing

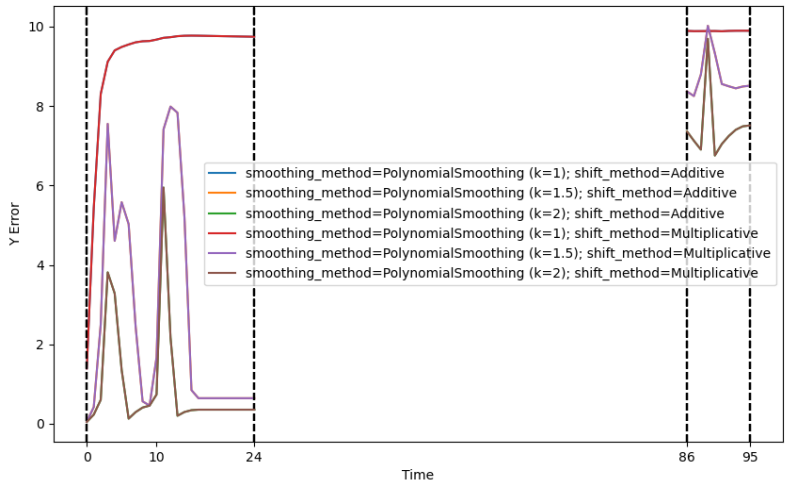

Figure: y dimension error with polynomial smoothing and varying k and shift method

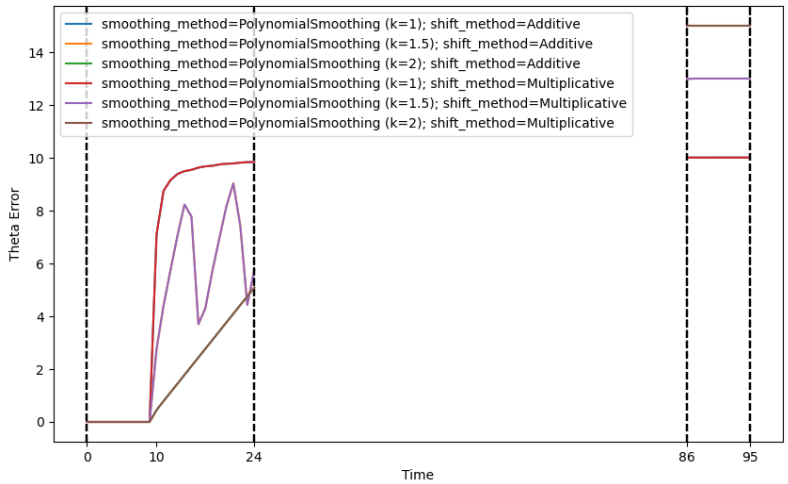

Figure: θ dimension error with polynomial smoothing and varying k and shift method

Olivier

RatSLAM 2.0 Algorithm Testing

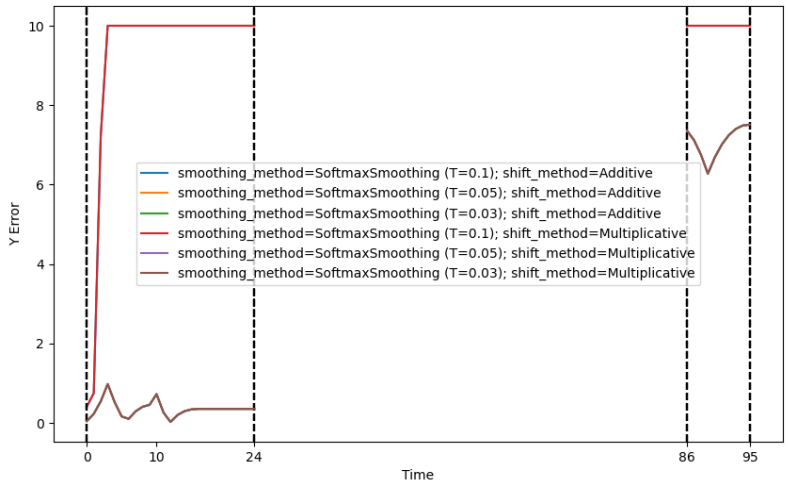

Figure: y dimension error with softmax smoothing and varying k and shift method

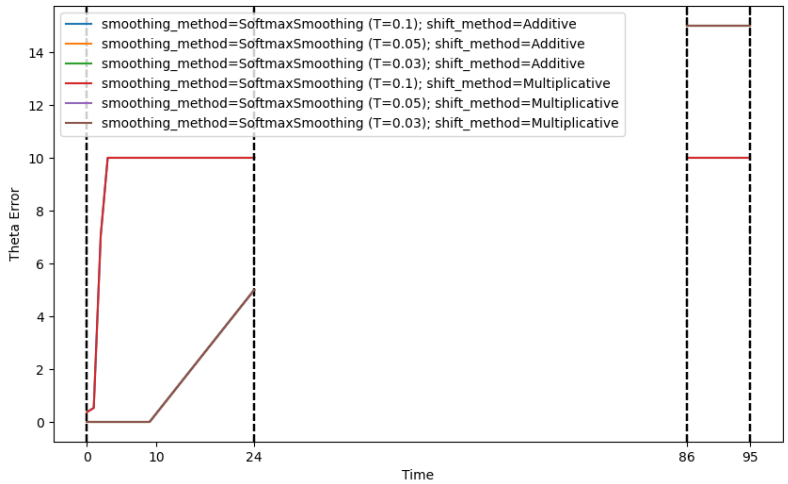

Figure: θ dimension error with softmax smoothing and varying k and shift method

Olivier

Critical Implementation Insights

Key Results

- **Kidnapping Recovery**: No method succeeded
- **Storage Strategy**: Hard/soft equivalent for pose
- **Merging Criticality**: Multiplicative $\gg$ Additive

Architectural Implications

- Fixed-point preservation crucial for pose stability
- Sensory-hippocampal coupling needs separate optimization
- Full hyperparameters: Table 3

Olivier

References I

Sarthak Chandra, Sugandha Sharma, Rishidev Chaudhuri, and Ila Fiete.
Episodic and associative memory from spatial scaffolds in the hippocampus.
Nature, 638(8051):739–751, Feb 2025.

Michael Milford, Gordon Wyeth, and David Prasser.
Ratslam: a hippocampal model for simultaneous localization and mapping.
International Conference on Robotics and Automation, Jul 2004.

M.J. Milford and G.F. Wyeth.
Mapping a suburb with a single camera using a biologically inspired slam system.
IEEE Transactions on Robotics, 24(5):1038–1053, Oct 2008.

References II

Taahaa Mir, Peipei Yao, Kateri Duranceau, and Isabeau Prémont-Schwarz.
Are grid cells hexagonal for performance or by convenience?
Montreal AI and Neuroscience Conference, Oct 2024.

J. Tapson and A. van Schaik.
Learning the pseudoinverse solution to network weights.
Neural Networks, 45:94–100, Sep 2013.

Sebastian Thrun, Wolfram Burgard, and Dieter Fox.
Probabilistic robotics.
Mit Press, Cambridge, Mass., 2010.

References III

Konstantinos Voudouris, Ibrahim Alhas, Wout Schellaert, Matthew Crosby, Joel Holmes, John Burden, Niharika Chaubey, Niall Donnelly, Ben Slater, Matteo Mecattaf G, Matishalin Patel, Marta Halina, José Hernández-Orallo, and Lucy G. Cheke.
The animal-ai environment: A virtual laboratory for comparative cognition and artificial intelligence research.
arXiv preprint arXiv:2312.11414, 2024.

Appendix I

To be an effective scaffold, hippocampal cell activation must be sparse, i.e. the expected percentage of non-zero hippocampal cells active in a fixed point should be low [1]. Let $\psi \in [0, 1]$ denote a target sparsity, with $\psi = 1$ corresponding to the case where no neurons are active.
Sparsity in hippocampal cell activation can be achieved with two methods. The first one, used in VectorHaSH [1], is by sparsely initializing the projection matrix W_{hg} from grid to hippocampal states and using a rectifying function with computing h:

$$(W_{hg})_{ij} \sim \mathcal{N}(0, 1)\text{Bern}(\gamma) \approx \mathcal{N}(0, \gamma) \tag{3}$$

with $\gamma \in [0, 1]$ controlling the sparsity of the matrix. Then h is computed by:

$$h_i = \text{ReLU}\left(\sum_{j=1}^{N_g}(W_{hg})_{ij}g_j - \theta\right) \tag{4}$$

Appendix II

Since there will only be M nonzero entries of g when g is a concatenation of one-hot vectors, we have that

$$h_j \sim \text{ReLU}(\mathcal{N}(0, M\gamma) - \theta) \tag{5}$$

Therefore we wish to find θ such that $P(h_j \leq 0) = \psi$. By using the definition of the normal distribution and standardizing, we have that

$$\psi = \int_{-\infty}^{\frac{\theta}{\sqrt{M\gamma}}} f_Z(z)dz \tag{6}$$

where $Z \sim \mathcal{N}(0, 1)$. Hence,

$$\theta = \sqrt{M\gamma}\Phi^{-1}(\psi) \quad \text{or} \quad \theta = \sqrt{M\gamma}\left(\frac{1 + \text{Erf}\left(\frac{\psi}{\sqrt{2}}\right)}{2}\right) \tag{7}$$

Appendix III

The second method, that we propose, is to draw entries the projection matrix from a normal distribution with nonzero mean. Given a desired variance σ^2, M modules and a desired ψ sparsity, we have

$$(W_{hg})_{ij} \sim \mathcal{N}(\mu, \sigma^2) \tag{8}$$

Hence,

$$h_j \sim \text{ReLU}\left(\mathcal{N}(M\mu, M\sigma^2)\right) \tag{9}$$

Therefore, to achieve ψ sparsity, we solve for μ:

$$\psi = \int_{-\infty}^{\frac{-M\mu}{\sqrt{M}\sigma}} f_Z(z)dz \tag{10}$$

Hence

$$\mu = -\frac{\sigma}{\sqrt{M}}\Phi^{-1}(\psi) \quad \text{or} \quad \mu = -\frac{\sigma}{\sqrt{M}}\left(\frac{\text{Erf}\left(1+\frac{\psi}{\sqrt{2}}\right)}{2}\right) \tag{11}$$

Appendix IV

Suppose a random variable X is defined $X \sim \text{ReLU}\left(\mathcal{N}(\mu, \sigma^2)\right)$. Note that this is sufficient to handle the case that $X \sim \text{ReLU}\left(\mathcal{N}(\mu, \sigma^2) - \Theta\right)$ since $\text{ReLU}\left(\mathcal{N}(\mu, \sigma^2) - \Theta\right) = \text{ReLU}\left(\mathcal{N}(\mu - \theta, \sigma^2)\right)$.
Then by definition,

$$E[X] = \int_{-\infty}^{\infty} x f_X(x) dx = \int_{-\infty}^{\infty} \text{ReLU}(x) \frac{1}{\sqrt{2\pi}\sigma} e^{-(\frac{x-\mu}{\sqrt{2}\sigma})^2} dx = \int_{0}^{\infty} x \frac{1}{\sqrt{2\pi}\sigma} e^{-(} \tag{12}$$

Standardizing, we have that

$$E[X] = \int_{-\frac{\mu}{\sigma}}^{\infty} \frac{1}{\sqrt{2\pi}} (\sigma z + \mu) e^{-\frac{z^2}{2}} dz \tag{13}$$

Appendix V

Then by using standard integration techniques we obtain that

$$E[x] = \mu\Phi^{-1}\left(\frac{\mu}{\sigma}\right) + \frac{\sigma}{\sqrt{2\pi}}e^{-(\frac{\mu}{\sqrt{2}\sigma})^2} = \frac{\mu}{2}\text{Erf}\left(1 + \frac{\frac{\mu}{\sigma}}{\sqrt{2}}\right) + \frac{\sigma}{\sqrt{2\pi}}e^{-(\frac{\mu}{\sqrt{2}\sigma})^2} \tag{14}$$

Table: Hyperparameters for bidirectional iterative pseudoinverse learning.

ϵ_{sh}	0.1
ϵ_{hs}	0.1
M_{sh}	$1 \cdot$ input neurons
M_{hs}	$1 \cdot$ input neurons

Appendix VI

Table: Scaffold hyperparameters for hippocampal-sensory layer testing.

$\boldsymbol{\lambda}$	[(3,3,3),(4,4,4)]
N_h	400
Initialization method	Sparsity
ψ	0.9
γ	0.1

Table: Hyperparameters for AnimalAI tests.

Appendix VII

$\boldsymbol{\lambda}$	[(5,5,5),(8,8,8)]
N_h	600
Initialization method	Sparsity
ψ	0.1
γ	0.1
h-s layer	Iterative pseudoinverse
ϵ_{sh}	0.1
ϵ_{hs}	0.1
M_{sh}	$1 \cdot$ input neurons
M_{hs}	$1 \cdot$ input neurons
σ_{xy}	0.3
σ_{θ}	0.3
Global inhibition constant	0.004
δ_{γ}	1