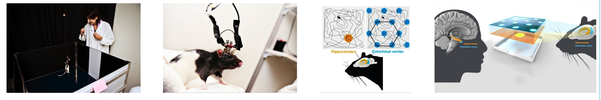
Learning Grid Cells as Vector Representation of Self-position Coupled with Matrix Representation of Self-motion

Ruiqi Gao[1,*], Jianwen Xie[2,*], Song-Chun Zhu[1], Ying Nian Wu[1] (*equal contribution)
[1] University of California, Los Angeles, [2] Hikvision Research Institute

Abstract

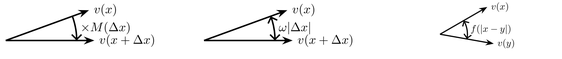
This paper proposes a representational model for grid cells. In this model, the 2D self-position of the agent is represented by a high-dimensional vector, and the 2D self-motion or displacement of the agent is represented by a matrix that transforms the vector. Each component of the vector is a unit or a cell. The model can learn hexagon patterns of grid cells from simulated trajectories, and it is capable of error correction, path integral and path planning.

Background: grid cells



The activity of neurons are recorded when the rat is moving within a square region. Each place cell fires at a particular location, but each grid cell fires at multiple locations that form a hexagon grid.

Representational model of grid cells



(1) Vector-matrix multiplication (2) Magnified local isometry (3) Global adjacency kernel

Sub-model 1 motion algebra: vector-matrix multiplication

Suppose at a position x, the self-motion or one-step displacement is Δx. The agent moves to $x + \Delta x$ after one step. We assume that

$$v(x + \Delta x) = M(\Delta x)v(x), \tag{1}$$

$v(x)$: the vector representation of the self-position x
$M(\Delta x)$: the matrix representation of the self-motion Δx, which can be parametrized as a polynomial function of Δx.

$$\text{Motion}: \quad \begin{array}{ccc} x_t & \xrightarrow{+\Delta x} & x_{t+1} \\ \downarrow & & \downarrow \\ v(x_t) & \xrightarrow{M(\Delta x)\times} & v(x_{t+1}) \end{array} \tag{2}$$

- Disentangled blocks or modules: assume that $M(\Delta x)$ is block diagonal, i.e.,

$$v = (v^{(k)}, k = 1, ..., K), \tag{3}$$
$$v^{(k)}(x + \Delta x) = M^{(k)}(\Delta x)v^{(k)}(x) \tag{4}$$

Sub-model 2 local geometry: magnified local isometry

We assume that for each block,

$$\langle v^{(k)}(x), v^{(k)}(x + \Delta x)\rangle = d(1 - \alpha_k|\Delta x|^2), \tag{5}$$

for all x and Δx such that $\alpha_k|\Delta x|^2 \le c$. α_k can be either designed or learned.

- $1 - \alpha_k|\Delta x|^2$ is a second order Taylor expansion of a function $f(r)$ such that $f(0) = 1$, $f'(0) = 0$, i.e., 0 is the maximum, and $f''(0) = -2\alpha_k$.
- Let $\Delta\theta$ be the angle between $v^{(k)}(x)$ and $v^{(k)}(x + \Delta x)$. $\omega_k = \sqrt{2\alpha_k}$, defining the metric of block k, we have

$$\textbf{Magnified local isometry}: \Delta\theta = \omega_k|\Delta x|, \tag{6}$$

- Rotation: $\|v^{(k)}(x)\|$ is a constant, $M^{(k)}(\Delta x)$ is an orthogonal matrix, and the self-motion is a rotation in the d-dimensional space. $\omega_k|\Delta x|$: angular speed.

Vector rotates fast, back to itself $\rightarrow$ periodic pattern

- Projection: For a vector $v^{(k)}$, we can decode its position by projecting it onto the 2D sub-manifold: $\hat{x} = \arg\max_x \langle v^{(k)}, v^{(k)}(x)\rangle$.
- Error correction: for $\omega_k = \sqrt{2\alpha_k} \gg 1$, the magnification offers error correction.

Sub-model 3 global geometry: adjacent kernel

We assume that for the whole vector,

$$\langle v(x), v(y)\rangle = \sum_{k=1}^{K} \langle v^{(k)}(x), v^{(k)}(y)\rangle = (Kd)f(|x - y|), \tag{7}$$

$f(r)$ is the adjacency kernel that decreases monotonically.

- Let θ be the angle between $v(x)$ and $v(y)$, and we have

$$\textbf{Global adjacency}: \cos\theta = f(|x - y|). \tag{8}$$

- $(v(x), \forall x)$ forms a global codebook for x. $h(x) = \langle v, v(x)\rangle$ gives us the heat map to decode the position of v uniquely: $\hat{x} = \arg\max_x \langle v, v(x)\rangle$.
- Let $V = (v(x), \forall x)$, δ_x is a one-hot representation of a position x, we have the following encoding and decoding process:

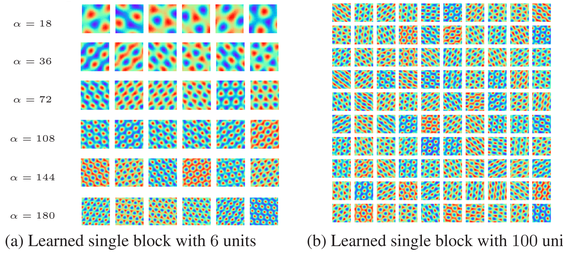
$$\text{Localization}: \quad \begin{array}{ccc} v & \xrightarrow{V^\top\times} & h \quad \text{(heat map and decoding to } \delta_x\text{)} \\ \delta_x & \xrightarrow{V\times} & v(x) \quad \text{(encoding)} \end{array} \tag{9}$$

Learning representation

We can learn $(v(x), \forall x)$ and $(M(\Delta x), \forall \Delta x)$ (or the β coefficients that parametrize M) by minimizing the following loss functions.
Model 1 loss: $L_1 = \mathrm{E}_{x,\Delta x}\left[\|v(x + \Delta x) - M(\Delta x)v(x)\|^2\right]$,
Model 2 loss: $L_{2,k} = \mathrm{E}_{x,\Delta x}\left[(\langle v^{(k)}(x), v^{(k)}(x + \Delta x)\rangle - (d(1 - \alpha_k|\Delta x|^2))^2\right]$,
Model 3 loss: $L_3 = \mathrm{E}_{x,y}\left[((Kd)f(|x - y|) - \langle v(x), v(y)\rangle)^2\right]$,
Regularization loss: $L_0 = \sum_{i=1}^{Kd}(\mathrm{E}_x[v_i(x)^2] - 1)^2$.
The total loss function is a linear combination of the above losses.
Simulated trajectories are used to learn the system.

Exp 1 learning single blocks: hexagon patterns and metrics

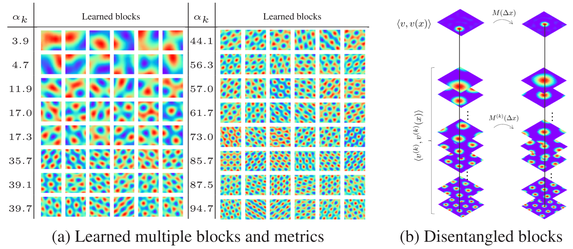
We first learn a single block with fixed α_k by minimizing $L_1 + \lambda_2 L_{2,k} + \lambda_0 L_0$. The units within a block have patterns with similar scale and arrangement, yet different phases. Learned each unit (or dimension) of $v(x)$ over x:



(a) Learned single block with 6 units (b) Learned single block with 100 units

Exp 2 learning multiple blocks and metrics

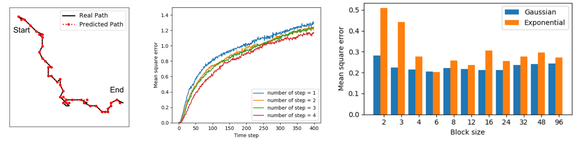
We learn multiple blocks by minimizing $L_1 + \lambda_2 L_2 + \lambda_3 L_3 + \lambda_0 L_0$. Instead of manually assigning α_k, we learn α_k by gradient descent, simultaneously with v and M. A Gaussian kernel with $\sigma = 0.08$ is used for the global adjacency measure $f(|x - y|)$.



(a) Learned multiple blocks and metrics (b) Disentangled blocks

Exp 3 path integral

Path integral (also referred to as dead-reckoning) is the task of inferring the self-position based on self-motion (e.g., imagine walking in a dark room).
Input: initial position x_0 and motion sequences $\{\Delta x_1, ..., \Delta x_T\}$.
Output: the prediction of one's current position x_T.
Algorithm: Encode the initial position x_0 as $v(x_0)$. Then, the hidden vector $v(x_T)$ at time T can be predicted as: $v(x_T) = \prod_{t=T}^{1} M(\Delta x_t)v(x_0)$. We can then decode x_T from $v(x_T)$.



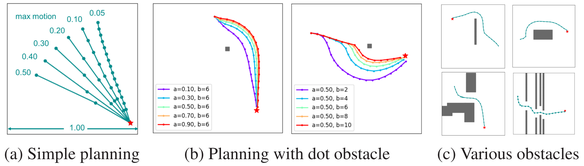
(a) Predicted path (b) MSE over time step (c) Choice of block sizes and kernels

Exp 4 path planning

Input: the starting position x_0, the target position y and some obstacles $\{z_i\}_{i=1}^m$.
Output: the planned path to the target.
Algorithm: Let $v_0 = v(x_0)$. Δ is the set of allowable displacements Δx. Then the algorithm iterates

$$\Delta x_t = \arg\max_{\Delta x \in \Delta} \left[\langle v(y), M(\Delta x)v_t\rangle - a\langle v(z), M(\Delta x)v_t\rangle^b\right], \tag{10}$$
$$v_t = M(\Delta x_t)v_{t-1}, \tag{11}$$

where a and b are the scaling and annealing parameters. The system is learned with exponential kernel ($\sigma = 0.3$) as global adjacency.



(a) Simple planning (b) Planning with dot obstacle (c) Various obstacles

Reference

[1] Banino, Andrea, et al. "Vector-based navigation using grid-like representations in artificial agents." Nature 557.7705 (2018): 429.
[2] Cueva, Christopher J., and Xue-Xin Wei. "Emergence of grid-like representations by training recurrent neural networks to perform spatial localization." *ICLR*(2018).