Julia で学ぶ計算論的神経科学

山本 拓都

2023 年 5 月 8 日

目次

第 1 章

はじめに

第 2 章

神経細胞のモデル

第 3 章

シナプス伝達のモデル

第 4 章

神経回路網の演算処理

第 5 章

局所学習則

5.1 自己組織化マップと視覚野の構造

視覚野にはコラム構造が存在する．こうした構造は神経活動依存的な発生 (activity dependent development) により獲得される．本節では視覚野のコラム構造を生み出す数理モデルの中で，**自己組織化マップ (self-organizing map)** [?], [?] を取り上げる．

自己組織化マップを視覚野の構造に適応したのは [?] [?] などの研究である．視覚野マップの数理モデルとして自己組織化マップは受容野を考慮しないなどの簡略化がなされているが，単純な手法にして視覚野の構造に関する良い予測を与える．他の数理モデルとしては自己組織化マップと発想が類似している **Elastic net** [?] [?] [?]　(ここでの Elastic net は正則化手法としての Elastic net regularization とは異なる) や受容野を明示的に設定した [?], [?] などのモデルがある．総説としては [?], [?] ，数理モデル同士の関係については [?] が詳しい．

自己組織化マップでは「抹消から中枢への伝達過程で損失される情報量」，および「近い性質を持ったニューロン同士が結合するような配線長」の両者を最小化するような学習が行われる．包括性 (coverage) と連続性 (continuity) のトレードオフとも呼ばれる [?]　(Elastic net は両者を明示的に計算し，線形結合で表されるエネルギー関数を最小化する．Elastic net は本書では取り扱わないが，MATLAB 実装が公開されている https://faculty.ucmerced.edu/mcarreira-perpinan/research/EN.html) ．連続性と関連する事項として，近い性質を持つ細胞が脳内で近傍に存在する現象があり，これを**トポグラフィックマッピング (topographic mapping)** と呼ぶ．トポグラフィックマッピングの数理モデルの初期の研究としては [?] [?] [?] などがある．

発生の数理モデルに関する総説 [?], [?]

5.1.1 単純なデータセット

SOM における n 番目の入力を $\mathbf{v}(t) = \mathbf{v}_n \in \mathbb{R}^D (n = 1, \ldots, N)$, m 番目のニューロン $(m = 1, \ldots, M)$ の重みベクトル (または活動ベクトル, 参照ベクトル) を $\mathbf{w}_m(t) \in \mathbb{R}^D$ とする [?]．また，各ニューロンの

物理的な位置を $\mathbf{x}_m$ とする．このとき，$\mathbf{v}(t)$ に対して $\mathbf{w}_m(t)$ を次のように更新する．

まず，$\mathbf{v}(t)$ と $\mathbf{w}_m(t)$ の間の距離が最も小さい (類似度が最も大きい) ニューロンを見つける．距離や類似度としてはユークリッド距離やコサイン類似度などが考えられる．

$$[\text{ユークリッド距離}] : c = \underset{m}{\operatorname{argmin}} \left[\|\mathbf{v}(t) - \mathbf{w}_m(t)\|^2 \right] \tag{5.1}$$

$$[\text{コサイン類似度}] : c = \underset{m}{\operatorname{argmax}} \left[\frac{\mathbf{w}_m(t)^\top \mathbf{v}(t)}{\|\mathbf{w}_m(t)\| \|\mathbf{v}(t)\|} \right] \tag{5.2}$$

この，c 番目のニューロンを**勝者ユニット (best matching unit; BMU)** と呼ぶ．コサイン類似度において，$\mathbf{w}_m(t)^\top \mathbf{v}(t)$ は線形ニューロンモデルの出力となる．このため，コサイン距離を採用する方が生理学的に妥当であり SOM の初期の研究ではコサイン類似度が用いられている [**?**]．しかし，コサイン類似度を用いる場合は $\mathbf{w}_m$ および $\mathbf{v}$ を正規化する必要がある．ユークリッド距離を用いると正規化なしでも学習できるため，SOM を応用する上ではユークリッド距離が採用される事が多い．ユークリッド距離を用いる場合，$\mathbf{w}_m$ は重みベクトルではなくなるため，活動ベクトルや参照ベクトルと呼ばれる．ここでは結果の安定性を優先してユークリッド距離を用いることとする．

こうして得られた c を用いて $\mathbf{w}_m$ を次のように更新する．

$$\mathbf{w}_m(t+1) = \mathbf{w}_m(t) + h_{cm}(t)[\mathbf{v}(t) - \mathbf{w}_m(t)] \tag{5.3}$$

ここで $h_{cm}(t)$ は近傍関数 (neighborhood function) と呼ばれ，c 番目と m 番目のニューロンの距離が近いほど大きな値を取る．ガウス関数を用いるのが一般的である．

$$h_{cm}(t) = \alpha(t) \exp\left(-\frac{\|\mathbf{x}_c - \mathbf{x}_m\|^2}{2\sigma^2(t)} \right) \tag{5.4}$$

ここで $\mathbf{x}$ はニューロンの位置を表すベクトルである．また，$\alpha(t), \sigma(t)$ は単調に減少するように設定する．[*1]

```
using Random, PyPlot, ProgressMeter
using PyPlot: matplotlib
rc("font", family="Arial")
```

ToDo: dims を v, w で修正

```
# inputs
Random.seed!(1234);
σv, σw = 0.1, 0.05
```

[*1] Generative topographic map (GTM) を用いれば $\alpha(t), \sigma(t)$ の縮小は必要ない．また，SOM と GTM の間を取ったモデルとして S-map がある．

```
dims = 2  # dims of inputs and neurons
num_v = 300 # num of inputs
num_blobs = 5 # num. cluster of dataset
num_w_sqrt = 15 # must be int
num_w = num_w_sqrt^2
init_w = σw*randn(num_w, dims);
```

```
# 単位円上に等間隔にならんだクラスターによるtoy datasetを作成する
function make_blobs(num_samples, num_blobs, dims, σ)
    n = Int(num_samples/num_blobs) # number of samples in each
    data = vcat([σ*randn(n, dims) .+ [cos(i/num_blobs*2π), ↵
        sin(i/num_blobs*2π)]' for i in 0:num_blobs-1]...)
    label = repeat(1:num_blobs, inner=n)
    return data, label
end
```

```
v, v_labels = make_blobs(num_v, num_blobs, dims, σv);
```

```
function plot_som(v, w; vcolor="tab:blue", wcolor="tab:orange")
    num_w, dims = size(w)
    num_w_sqrt = Int(sqrt(num_w))
    rw = reshape(w, (num_w_sqrt, num_w_sqrt, dims))
    scatter(v[:, 1], v[:, 2], s=10, color=vcolor)
    plot(rw[:, :, 1], rw[:, :, 2], "k", alpha=0.8); plot(rw[:, :, 1]', ↵
        rw[:, :, 2]', "k", alpha=0.8)
    scatter(w[:, 1], w[:, 2], s=10, color=wcolor) # w[i, j, 1]とw[i, j, ↵
        2]の点をプロット
end;
```

近傍関数 (neighborhood function) のための二次元ガウス関数を実装する．Winner ニューロンからの距離に応じて値が減弱する関数である．ここでは一つの入力に対して全てのニューロンの活動ベクトルを更新するということはせず，winner neuron の近傍のニューロンのみ更新を行う．つまり，更新においては global ではなく local な処理のみを行うということである (Winner neuron の決定には WTA による global な評価が必要ではあるが)．自己組織化マップのメインとなる関数を書く．ナイーブに実装する．この方法だと空間が球体やトーラスのように周期性を持つ場合にも適応できる．

```
function SOM(v, init_w; α0=1.0, σ0=6, T=500, dist_mat=nothing, ↵
    return_history=true)
    # α0: update rate, σ0 : width, T : training steps
    w = copy(init_w)
```

```
    num_w = size(init_w)[1]
    num_w_sqrt = Int(sqrt(num_w))
    num_v = size(v)[1]

    if return_history
        w_history = [copy(init_w)] # history of w
    end

    if dist_mat == nothing
        pos = hcat([[i, j] for i in 1:num_w_sqrt for j in 1:num_w_sqrt]...)
        dist_mat = hcat([sum((pos .- pos[:, i]) .^2, dims=1)' for i in ↵
            1:num_w]...); #'
    end

    @showprogress for t in 1:T
        α = α0 * (1 - t/T); # update rate
        σ = max(σ0 * (1 - t/T), 1); # decay from large to small (linearly ↵
            decreased, avoid zero)
        exp_dist_mat = exp.(-dist_mat / (2.0(σ^2)))
        exp_dist_mat ./= maximum(sum(exp_dist_mat, dims=1))
        # loop for the num_v inputs
        for i in 1:num_v
            dist = sum((v[i, :]' .- w).^2, dims=2) # distance between input ↵
                and neurons
            win_idx = argmin(dist)[1] # winner index
            # update the winner & neighbor neuron
            η = α * exp_dist_mat[win_idx, :]
            w[:, :] += η .* (v[i, :]' .- w)
        end
        if return_history
            append!(w_history, [copy(w)]) # save w
        end
    end
    if return_history
        return w_history
    else
        return w
    end
end;
```

今回のように 2 次元のみを扱う場合は winner neuron の周辺だけを slice で抜き出して重み更新する方が高速である.

```
# Gaussian mask for inputs
function gaussian_mask(sizex=9, sizey=9; σ=5)
    x, y = 0:sizex-1, 0:sizey-1
```

```
    X, Y = ones(sizey) * x', y * ones(sizex)'
    x0, y0 = (sizex-1) / 2, (sizey-1) / 2
    mask = exp.(-((X .- x0) .^2 + (Y .- y0) .^2) / (2.0(σ^2)))
    return mask ./ sum(mask)
end;
```

```
function SOM2d(v, init_w; α0=1.0, σ0=6, T=500, return_history=true)
    # α0: update rate, σ0 : width, T : training steps
    w = copy(init_w)
    num_w, dims = size(init_w)
    num_w_sqrt = Int(sqrt(num_w))
    num_v = size(v)[1]

    w_history = [copy(w)] # history of w

    w_2d = reshape(w, (num_w_sqrt, num_w_sqrt, dims))

    if return_history
        w_history = [copy(init_w)] # history of w
    end

    @showprogress for t in 1:T
        α = α0 * (1 - t/T); # update rate
        σ = max(σ0 * (1 - t/T), 1); # decay from large to small (linearly ↵
            decreased, avoid zero)
        wm = ceil(Int, σ)
        h = gaussian_mask(2wm+1, 2wm+1, σ=σ);
        # loop for the num_v inputs
        for i in 1:num_v
            dist = sum([(v[i, j] .- w_2d[:, :, j]).^2 for j in 1:dims]) # ↵
                distance between input and neurons
            win_idx = argmin(dist) # winner index
            idx = [max(1,win_idx[j] - wm):min(num_w_sqrt, win_idx[j] + wm) ↵
                for j in 1:2] # neighbor indices
            # update the winner & neighbor neuron
            η = α * h[1:length(idx[1]), 1:length(idx[2])]
            for j in 1:dims
                w_2d[idx..., j] += η .* (v[i, j] .- w_2d[idx..., j])
            end
        end
        if return_history
            w = reshape(w_2d, (num_w, dims))
            append!(w_history, [copy(w)]) # save w
        end
    end
```

```
    if return_history
        return w_history
    else
        w = reshape(w_2d, (num_w, dims))
        return w
    end
end;
```

```
w_history = SOM(v, init_w, α0=2, σ0=10, T=100);
#w_history = SOM2d(v, init_w, α0=2, σ0=10, T=100);
```

青点が **v**，オレンジの点が **w** である．黒線はニューロン間の位置関係を表す (これは Weight unfolding diagrams と呼ばれる)．下段のヒートマップは **w** の一番目の次元を表す．学習が進むとともに近傍のニューロンが近い活動ベクトルを持つことがわかる．

```
cm = get_cmap(:Reds)
vcolors = cm.(v_labels / num_blobs);
```

```
figure(figsize=(6, 4))
idx = [1, 50, 100]
for i in 1:length(idx)
    wh = w_history[idx[i]]
    subplot(2,length(idx),i)
    title("Epoch : "*string(idx[i]))

    if i == 1
        ylabel("Weight unfolding\n in data space")
    end
    plot_som(v, wh, vcolor=vcolors);
    subplot(2,length(idx),i+length(idx))

    if i == 1
        ylabel("1st dim. weight")
    end
    imshow(reshape(wh[:, 1], (num_w_sqrt, num_w_sqrt)));
end
tight_layout()
```

unified distance matrix を描画する．隣接する要素とは位置の差の絶対値が 1 であることを利用する．

```
function Umatrix2d(w)
    num_w = size(w)[1]
```

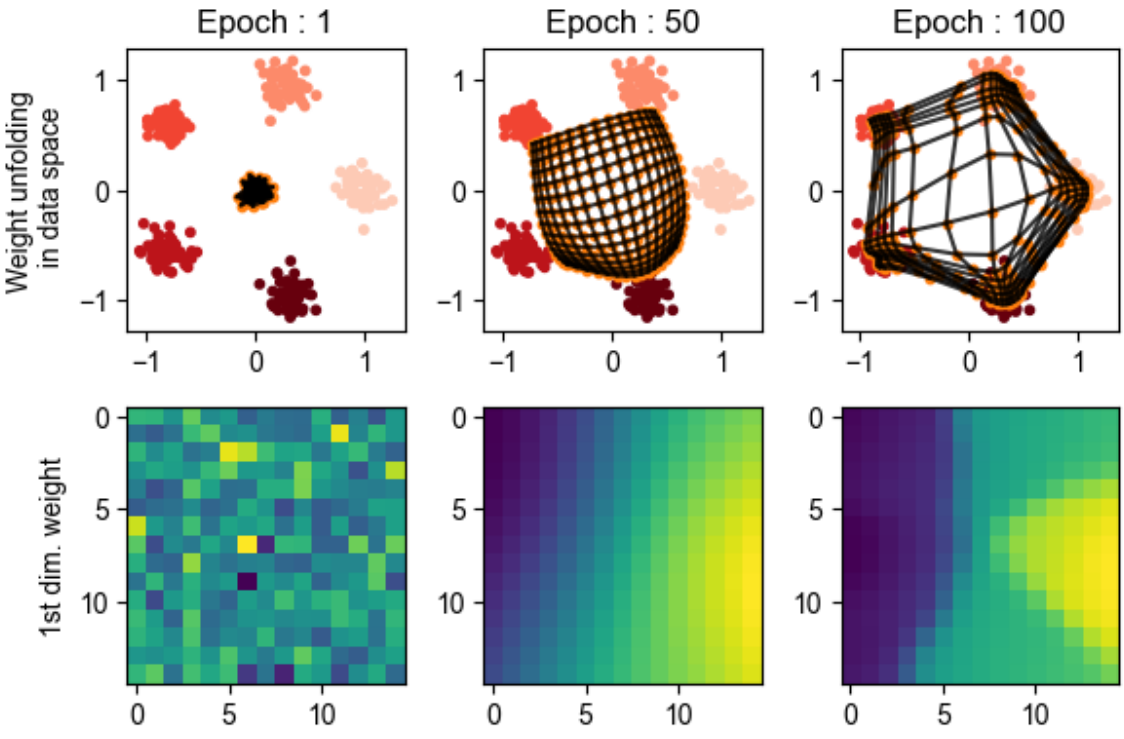

図 5.1 cell017.png

```
    num_w_sqrt = Int(sqrt(num_w))
    pos = hcat([[i, j] for i in 1:num_w_sqrt for j in 1:num_w_sqrt]...)
    abs_dist_mat = hcat([sum(abs.(pos .- pos[:, i]), dims=1)' for i in ↵
        1:num_w]...)
    adj_indices = [findall(x -> x == 1, abs_dist_mat[i, :]) for i in ↵
        1:num_w] # adjacent indices
    U = [sqrt(sum((w[adj_indices[i], :] .- w[i, :]') .^2) / ↵
        size(adj_indices[i])[1]) for i in 1:num_w]
    U = reshape(U, (num_w_sqrt, num_w_sqrt));
    return U
end
```

```
# find best matching unit
function find_bmu(v, w)
    num_v, dims = size(v)
    num_w = size(init_w)[1]
    num_w_sqrt = Int(sqrt(num_w))
```

```
    pos = hcat([[i, j] for i in 1:num_w_sqrt for j in 1:num_w_sqrt]...)
    mapped_vpos = zeros(num_v, dims);
    for i in 1:num_v
        dist = sum((v[i, :]' .- w).^2, dims=2) # distance between input and ↩
            neurons
        win_idx = argmin(dist)[1] # winner index
        mapped_vpos[i, :] = pos[:, win_idx]' .- 1
    end
    return mapped_vpos
end
```

```
U = Umatrix2d(w_history[end]);
```

```
mapped_vpos = find_bmu(v, w_history[end]);
```

複数の点が同じ位置に重なっていることに注意.

```
figure(figsize=(3,3))
title(L"$U$-matrix")
imshow(U, interpolation="bicubic")
scatter(mapped_vpos[:, 1], mapped_vpos[:, 2], color=vcolors, s=10)
tight_layout()
```

5.1.2 視覚野マップ

集合の直積を配列として返す関数 product と極座標を直交座標に変換する関数 pol2cart を用意する.

```
product(sets...) = hcat([collect(x) for x in ↩
    Iterators.product(sets...)]...)' # Array of Cartesian product of sets
pol2cart(θ, r) = r*[cos(θ), sin(θ)];
```

刺激と初期の活動ベクトルは [?] を参考に作成. 直積 product で全ての組の入力を作成する.

```
# generate stimulus
Random.seed!(1234);
Nx, Ny, NOD, NOR = 10, 10, 2, 12
dims = 5  # dims of inputs
l, r = 0.14, 0.2
```

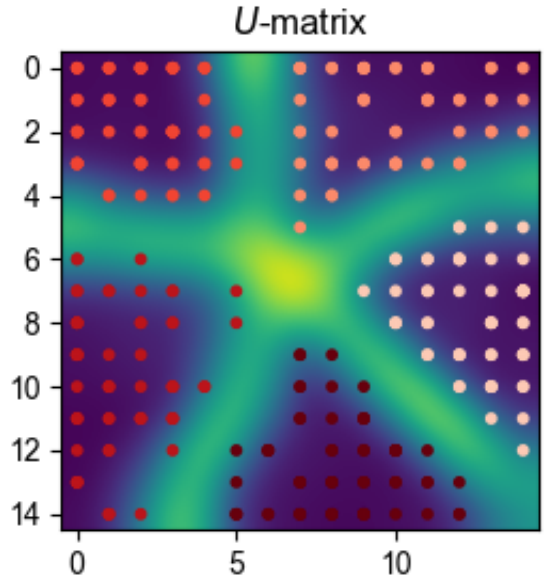

図 5.2 cell024.png

```
rx, ry = range(0, 1, length=Nx), range(0, 1, length=Ny)
rOD = range(-l, l, length=NOD)
rORθ = range(-π/2, π/2, length=NOR+1)[1:end-1]

# stimuli
v = product(rx, ry, rOD, rORθ, r)
rORxy = hcat(pol2cart.(2v[:, 4], v[:, 5])...)
v[:, 4], v[:, 5] = rORxy[1, :], rORxy[2, :];
v += (rand(size(v)...) .- 1) * 1e-5;
```

```
# initial neurons
num_w_sqrt = 64
num_w = num_w_sqrt^2
init_w = product(range(0, 1, length=num_w_sqrt), range(0, 1, ↵
    length=num_w_sqrt))
init_w += (rand(size(init_w)...) .- 1) * 0.05;
init_w = [init_w 2l*(rand(num_w) .- 0.5) hcat(pol2cart.(4π*(rand(num_w) .- ↵
    0.5), r*rand(num_w))...)']
#w = reshape(w, (num_w_sqrt, num_w_sqrt, dims));
```

```
w = SOM2d(v, init_w, α0=1.5, σ0=5.0, T=50, return_history=false); # faster
```

```
#w = SOM(v, init_w, α0=1.5, σ0=5.0, T=50, return_history=false);
```

描画用関数を実装する．w_history を用いてアニメーションを作成すると発達の過程が可視化されるが，これは読者への課題とする．

```
function plot_visual_maps(v, w)
    figure(figsize=(7, 6))
    subplot(2,2,1, adjustable="box", aspect=1); title("Retinotopic map")
    plot_som(v, w)

    num_w, dims = size(w)
    num_w_sqrt = Int(sqrt(num_w))
    rw = reshape(w, (num_w_sqrt, num_w_sqrt, dims))

    ax1 = subplot(2,2,2, adjustable="box", aspect=1); title("Ocular ↵
        dominance (OD) map")
    imshow(rw[:, :, 3], cmap="gray", origin="lower")

    ins1 = ax1.inset_axes([1.05,0,0.05,1])
    colorbar(cax=ins1, aspect=40, pad=0.08, shrink=0.6)
    ins1.text(0, -0.16, "Left", ha="left", va="center")
    ins1.text(0, 0.16, "Right", ha="left", va="center")

    subplot(2,2,3, adjustable="box", aspect=1); title("Contours of OD and ↵
        OR")
    ORmap = atan.(rw[:, :, 5], rw[:, :, 4]); # get angle of polar
    sizex, sizey = num_w_sqrt, num_w_sqrt
    x, y = 0:sizex-1, 0:sizey-1
    X, Y = ones(sizey) * x', y * ones(sizex)';
    contour(X, Y, ORmap, cmap="hsv")
    contour(X, Y, rw[:, :, 3], colors="k", levels=1)

    ax2 = subplot(2,2,4, adjustable="box", aspect=1); title("Orientation ↵
        (OR) angle map")
    imshow(ORmap, cmap="hsv", origin="lower")

    cm = get_cmap(:hsv)
    lines, colors = [], []
    for i in 1:9
        θ = (i-1)/8*π
        c, s = cos(θ), sin(θ)
        push!(lines, [(-c/2, 15-1.5i -s/2), (c/2, 15-1.5i + s/2)])
        push!(colors, cm(1/8*(i-1)))
    end

    ins2 = ax2.inset_axes([1,0,0.2,1])
```

```
    ins2.add_collection(matplotlib.collections.LineCollection(lines, ↵
        linewidths=3,color=colors))
    ins2.set_aspect("equal")
    ins2.axis("off")
    ins2.set_xlim(-1, 1); ins2.set_ylim(0, 15)

    tight_layout()
end;
```

```
plot_visual_maps(v, w)
```

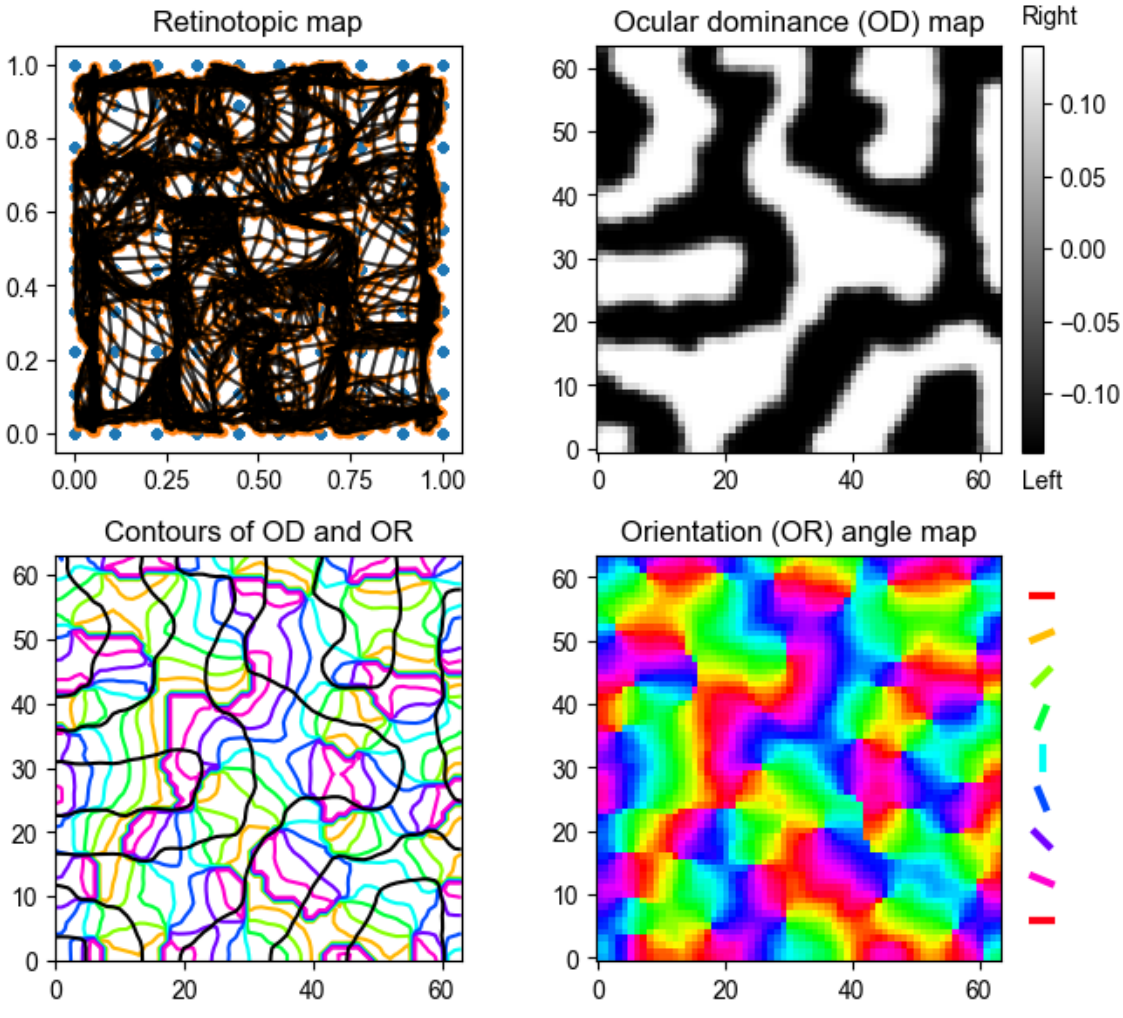

図 5.3 cell033.png

第 6 章

生成モデルとエネルギーベースモデル

第 7 章

貢献度分配問題の解決策

第 8 章

運動制御

第 9 章

強化学習

第 10 章

神経回路網によるベイズ推論