CUED - Engineering Tripos Part IIB 2023-2024 **Module Coursework**

Module	4G3	Title of report	4G3 Computational Neuroscience Assignment 1
Date submitted:	21 Feb 2024	Assessment for this module is ☑ 100% / ☐ 25% coursework of which this assignment forms 50 %	

UNDERGRADUATE and POST GRADUATE STUDENTS		
Candidate number:	5487G	☑ Undergraduate ☐ Post graduate

Feedback to the student ☐ **See also comments in the text**		Very good	**Good**	Needs improvmt
CONTENT	**Completeness, quantity of content:** Has the report covered all aspects of the lab? Has the analysis been carried out thoroughly?			
	Correctness, quality of content Is the data correct? Is the analysis of the data correct? Are the conclusions correct?			
	Depth of understanding, quality of discussion Does the report show a good technical understanding? Have all the relevant conclusions been drawn?			
	Comments:			
PRESENTATION	**Attention to detail, typesetting and typographical errors** Is the report free of typographical errors? Are the figures/tables/references presented professionally?			
	Comments:			

Marker: Date:

4G3 Computational Neuroscience Assignment 1

Candidate: 5487G

February 21, 2024

1 Introduction

The primary visual cortex (V1) is an area in the cerebral cortex which receives and processes visual stimulus from the Lateral Geniculate Nucleus (LGN). The V1 has been shown to possess orientation selectivity and contrast invariance [1]. The orientation-selective nature of the V1 could be explained by recurrent neural networks however, there are still many debates about how recurrent connections in the neurons are governed [2] [3] [4]. In this report, we will explore the 4 linear recurrent models outlined in the handout to explore the basic properties of recurrent models of the V1.

2 Question 1: Network Responses

Figure 1 shows the network response $r(t)$ of the 4 models at times $t = \{10, 50, 100, 200\}$ ms.

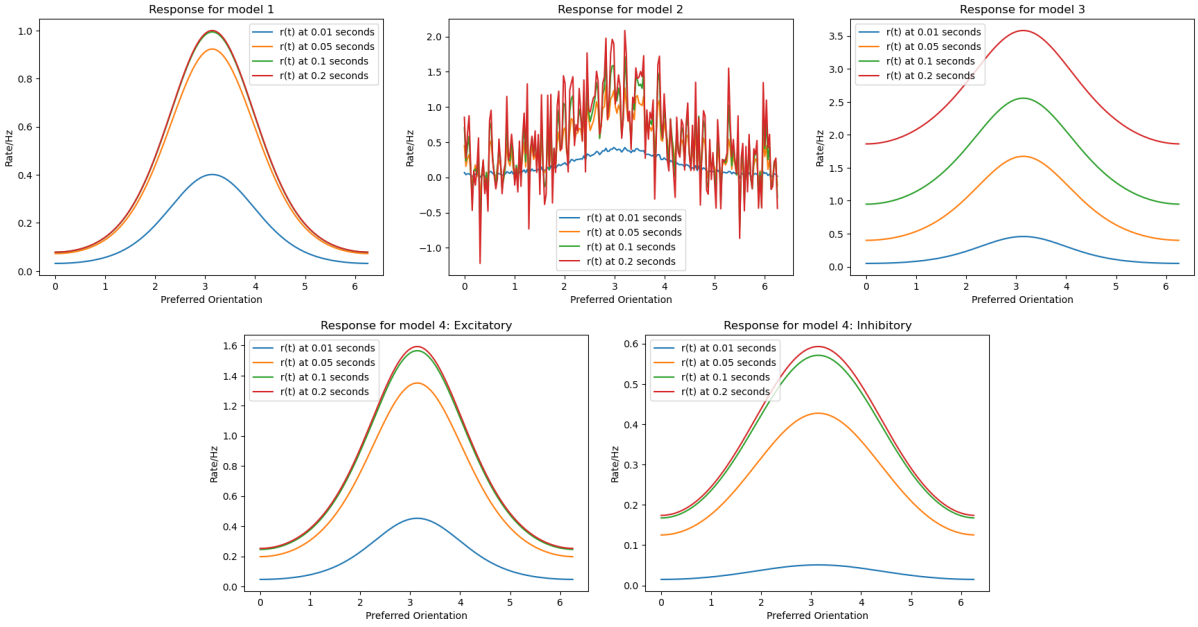

Figure 1: Network response for the 4 different models at different times. The bottom plots show the excitatory and inhibitory responses.

To analyse each model, we will explain the network's dynamic using the eigenvalues and eigenvectors of the recurrent connectivity matrix. Models 2 and 3 are symmetric models with a symmetric connectivity matrix, while the 4th model does not but does have each connection quadrant (EE, EI, IE, II) being symmetric.

From the handout, the linear recurrent model takes the form

$$\tau \frac{d\boldsymbol{r}}{dt} = -\boldsymbol{r} + W\boldsymbol{r} + B\boldsymbol{h}(\theta). \tag{1}$$

We assume B to be the identity matrix and thus can be dropped, and that $\boldsymbol{r}(0) = \boldsymbol{0}$. As W is a symmetric matrix, the eigenvectors of W are orthogonal which means we can represent the response as the sum of the weighted eigenvectors

$$\boldsymbol{r}(t) = \sum_{\mu=1}^{n} c_\mu(t)\boldsymbol{e}_\mu, \tag{2}$$

where $c_\mu(t)$ is the time-dependent coefficient. We could substitute equation 2 into 1 to solve for the coefficient. We would get

$$\tau \sum_{\mu=1}^{n} \frac{dc_\mu}{dt} \boldsymbol{e}_\mu = -\sum_{\mu=1}^{n} (1-\lambda_\mu)\, c_\mu(t) \boldsymbol{e}_\mu + \boldsymbol{h}(\theta), \tag{3}$$

with λ_μ being the eigenvalue of W. If we assume that each eigenvector is normalised (this is valid because are scaling the eigenvectors in equation 2), the orthonormality of the eigenvectors allows us to perform the dot product with one particular eigenvector, e_v, to get

$$\tau \frac{dc_v}{dt} = -(1-\lambda_v)\, c_v(t) + \boldsymbol{e_v} \cdot \boldsymbol{h}(\theta), \tag{4}$$

which is just a differential equation which has the solution [5]

$$c_v(t) = \frac{\boldsymbol{e}_v \cdot \boldsymbol{h}(\theta)}{1-\lambda_v} \left(1 - \exp\left(-\frac{t\,(1-\lambda_v)}{\tau}\right)\right). \tag{5}$$

Note that we have eliminated the initial condition term as $c_v(t) = 0$ because $\boldsymbol{r}(0) = \boldsymbol{0}$. To find the steady state of the system, we would sum each individual contribution to get

$$\boldsymbol{r_\infty} = \sum_{v=1}^{n} \frac{\boldsymbol{e}_v \cdot \boldsymbol{h}(\theta)}{1-\lambda_v} \boldsymbol{e}_v \tag{6}$$

From equation 5, we can imply that if the eigenvalue is greater than 1, then the exponential term diverges and causes the instability of the network. We could also infer that when the eigenvalue is less than 1, $c(t)$ would reach a steady state at $(\boldsymbol{e}_v \cdot \boldsymbol{h})/(1-\lambda_v)$ with a time constant $\tau/(1-\lambda_v)$. In other words, if lambda is close to 1 (and less than 1), the network would have a high steady state and respond slower. If lambda is not close to 1 (and less than 1), the network would have a lower steady state and respond quicker. From equation 6, if any eigenvalue is very small, the corresponding eigenvector term could therefore be disregarded. As we will see in the next sections, the results discussed here generalise well even with connectivity matrices which are not symmetric.

3 Question 2: Noisier Model 2

We observe that the symmetric random connectivity model is noisier than other models. This could be explained by looking at the eigenvalues and eigenvectors of each model. Figure 2 shows this.

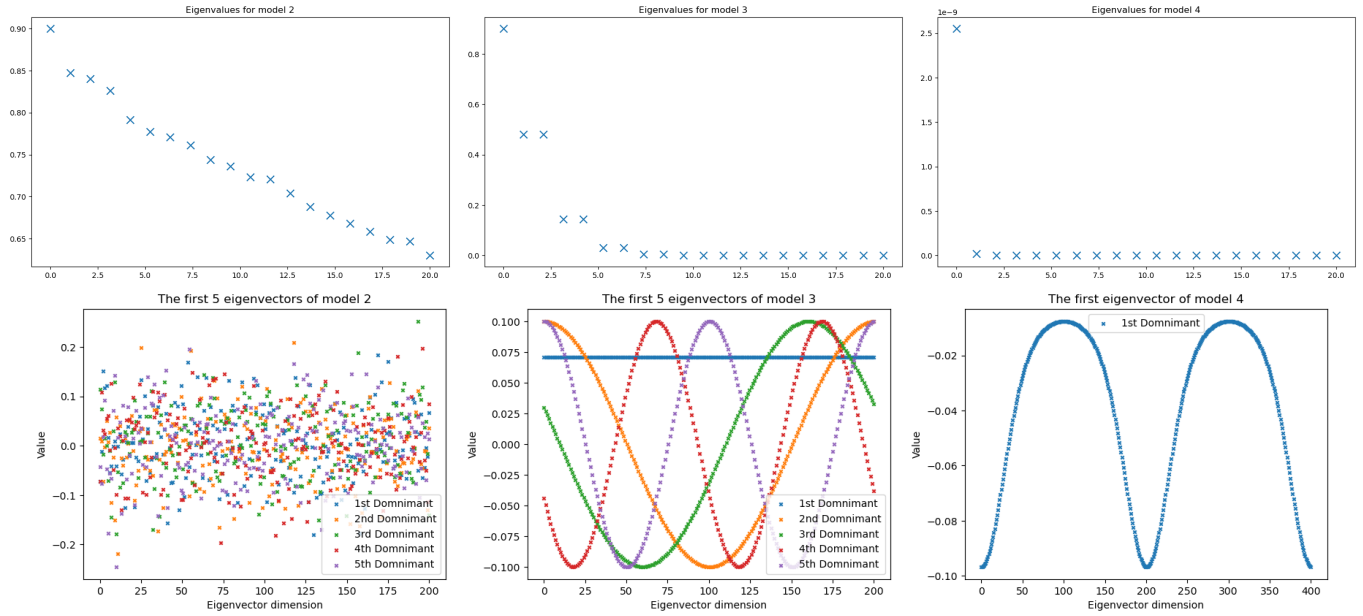

Figure 2: Top: The first 20 largest eigenvalues of each model. Bottom: The dominating eigenvectors of each model.

Model 1's response is smooth because the model lacks any recurrent connectivity and therefore the response is just the output of

$$\tau \frac{d\boldsymbol{r}}{dt} = \boldsymbol{h}(\theta) - \boldsymbol{r}, \tag{7}$$

and therefore takes the shape of the feed-forward input which is smooth. Model 2 has random eigenvalues and eigenvectors where no eigenvalue dominates which arise from the random connectivity. This results in the response

being noisy. Model 3 has a few dominating eigenvalues but because of the circular Gaussian connection, the eigenvectors are sinusoidal at different frequencies with a positive DC offset and thus result in a smooth response (as the weighted sum of a small number of sinusoids is smooth). Model 4 only has 1 dominating eigenvalue with the corresponding eigenvector being smooth for both excitatory and inhibitory dimensions. This results in the response of model 4 being smooth.

4 Question 3: Response Strength

From the analysis in the first section as well as figure 2, as model 2 and 3 have dominating eigenvalues which are a lot higher in magnitude than that of model 4, we could expect model 2 and 3 to have a higher steady state. Model 3 has a stronger steady state than model 2 because even though the spectral abscissa is the same, model 2 has inhibitory connections while model 3 does not. This naturally reduces the response of the model.

5 Question 4: Rate of Convergence

From the analysis in the first section as well as figure 2, models 2 and 3 have a relatively high dominating eigenvalue which therefore results in a longer time constant $\tau/(1-\lambda_v)$ than model 1 ($\lambda_v = 0$) and model 4. This leads to models 2 and 3 converging slower than model 1 and 4.

6 Question 5: Decoding Error

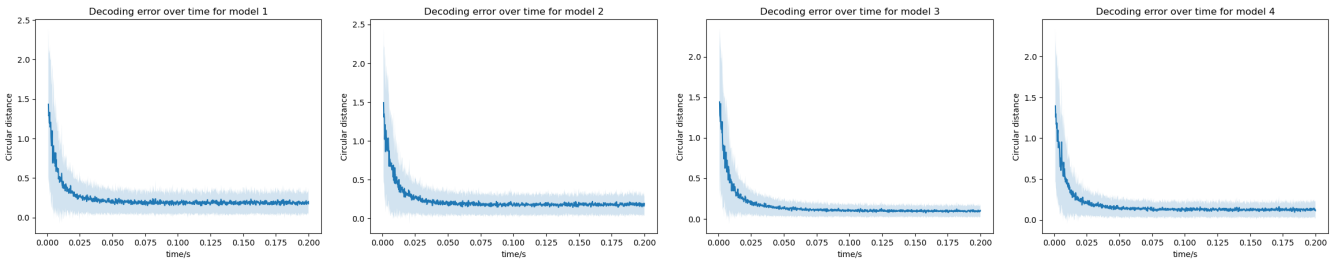

Figure 3: Decoding error over time for each model. Area in lighter blue shows ±1 standard deviation

	Average Decoding Error	Standard Deviation	SNR	Response Circular Variance
model 1	0.187	0.14	0.48	0.05
model 2	0.182	0.13	0.69	0.18
model 3	0.096	0.07	3.25	0.17
model 4	0.124	0.09	0.96	0.08

Table 1: Mean and standard deviation of the decoding error, and the signal-to-noise $\boldsymbol{r}(t)/\sigma\boldsymbol{\epsilon}(t)$ ratio and circular variance of the response for each model for $t = 0.2$s.

From figure 3 and table 1, in general, models 3 and 4 which have a ring structure, perform better than models 1 and 2 which do not have a ring structure. Models 1 and 2 do not differ much with model 2 being slightly better. This is expected as the random recurrent connectivity rule is Gaussian, it is equivalent to adding white noise to the signal at steady-state before further adding noise. Even though intuitively this must make decoding worse, the recurrent connectivity adds power to the output response which increases the signal-to-noise ratio overall. This results in a comparable performance between models 1 and 2.

Model 3 performs the best out of the 4 models. This is because of the high signal-to-noise ratio. A model can have a low decoding error if the steady-state response is high and/or the response circular variance is low.

7 Question 6: Model 4 High Alpha

By increasing α', we increase the magnitude of the eigenvalue. This would result in a stronger steady-state and intuitively decreases the signal-to-noise ratio which therefore improves the decoding accuracy. Figure 4 confirms this.

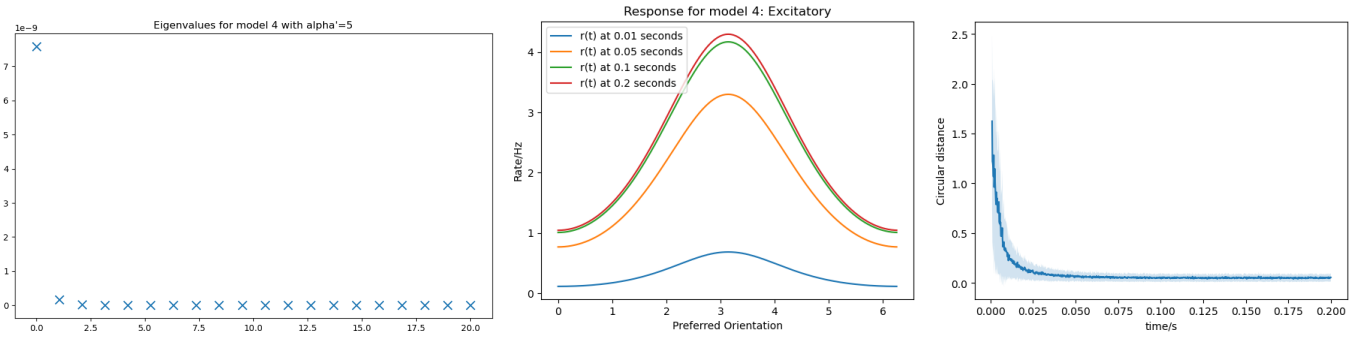

Figure 4: Plots for model 4 with $\alpha' = 5$. Left: Top 20 eigenvalues. Middle: Excitatory neuron response. Right: Decoding error over time

8 Question 7: Model 2 and 3 High Alpha

Unlike increasing α' in model 4, increasing α to 5 in models 2 and 3 will result in the maximum eigenvalue being more than 1. From our derivation in the first section, this will result in the model's response being unstable and growing exponentially. This is shown in figure 5 where the responses go up into the millions at $t = 0.1$s.

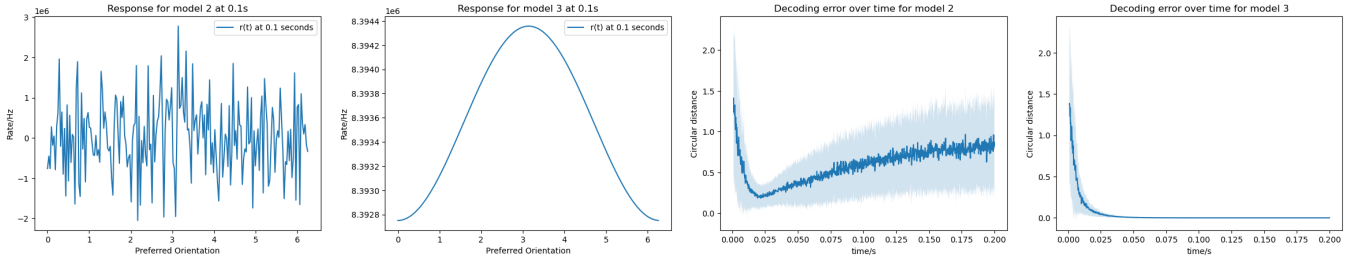

Figure 5: Plots for models 2 and 3 with $\alpha = 5$. From left to right, (1) response of model 2 after 0.1s. (2) response of model 3 after 0.1s (3) decoding error over time for model 2 (4) decoding error over time for model 3.

At first, the model will performance will improve as normal but because of the instability, after some time, model 2 will have the noise from the random connectivity being so high that the decoding accuracy decreases as well as the variance being larger.

On the other hand, the instability will cause model 3 to have the signal-to-noise ratio approaching 0 thus decreasing the average decoding error and variance to 0.

9 Question 8: Model 4 Input Weight

The input weight, B, represents the connection to one layer of V1 neurons from external excitatory neuron population or other V1 layers eg. layer 2/3 receivering signal from the LGN or Layer 4.

Therefore, to reduce the decoding error, we could connect this in a way which reduces the circular variance of the output or increase the steady-state response which will lower the SNR. To increase the steady-state response we could just scale B but that is undesirable. To decrease the circular variance, we could connect the external population with the inhibitory neurons in our model to inhibit regions which are not in the stimulus orientation. We could do this with the inverted circular Gaussian

$$V(\phi_i - \phi_j) = \exp\left(\frac{-\cos(\phi_i - \phi_j) - 1}{\kappa^2}\right), \tag{8}$$

The connection is therefore, $\begin{pmatrix} I_m \\ B_{inv} \end{pmatrix}$ where I_m is the identity matrix and B_{inv} is the connection matrix govern by equation 8 with $\alpha' = 0.9$. Figure 6 shows the results.

We can see that the inhibitory neurons inhibits neurons which are further away from the stimulus orientation. This results in a circular variance=0.07, SNR=0.49, and an average decoding error of 0.106 which despite the lower SNR, performs better than our base line model 4 (refer to table 1.

However, with this connection, the model rates goes into the negative region at steady state. This is not possible because rates, by definition, have to be positive. Furthermore, research suggests that connection are likely to be between the similarity of preferred orientation rather than dissimilarity [4][3].

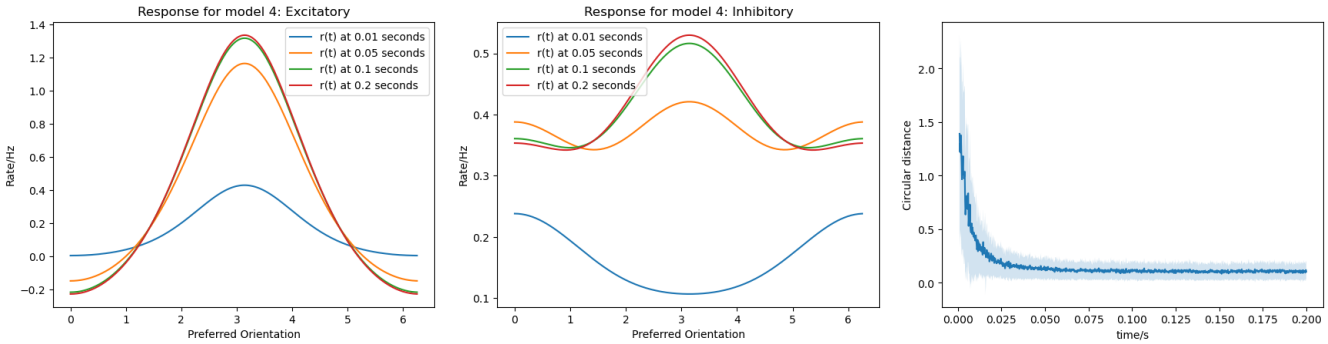

Figure 6: Left: Model's excitatory output. Middle: model inhibitory output. Right: Decoding error over time.

10 Question 9: Model 4 Output Weight

Again, we could increase the SNR to improve the model output. Scaling the matrix, C, would be able to achieve this. Furthermore, from figure 1, we could see that the inhibitory neurons also posses orientation selectivity very similarly to that of excitatory. This means that we could use $C = (I_m \ I_m)$ which will combine both excitatory and inhibitory neuron output and reduce SNR. From our simulation, wereceived an average decoding error of 0.95 which is an improvement from the baseline model. However, this is undesirable due to the violation of Dale's law because the output of the inhibitory neurons would be excitatory.

A better model is therefore one which is similar to the one in the previous section $C = (I_m \ -B_m)$. However, from the simulation, we got an average decoding error of 0.121 which is only slightly better than our baseline model. Figure 7 shows both simulations.

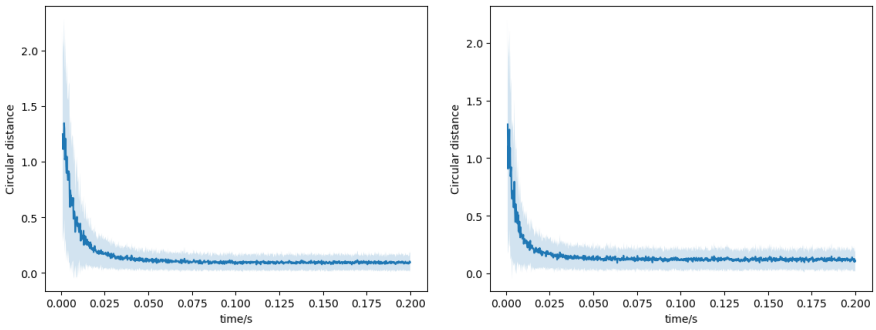

Figure 7: Model 4 decoding error over time. Left: $C = (I_m \ I_m)$. Right: $C = (I_m \ -B_m)$

11 Question 10: Model 4 Increasing Alpha

Model 4 could be improved by constantly increasing α'. However, this is not deistable because at very α', the model response rate will be very high which is not biologically possible. Another catch is that, our model is currently in a very tight balance as in both excitatory and inhibitory connections has the exact same value of α'. Once we breaks this balance, the model goes into instability. This is unlikely to be the case biologically. Figure 8 shows that the response diverging unstable with time with excitatory neurons having $\alpha' = 1000$ and inhibitory having $\alpha' = 999$ which is 0.1% difference. This effects worsen with more difference in α'.

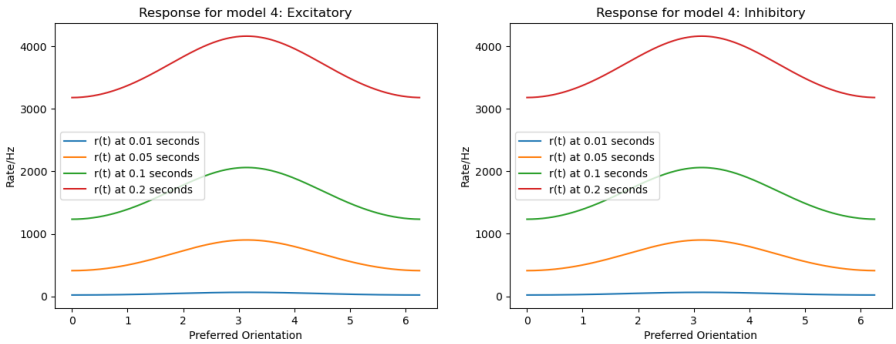

Figure 8: Excitatory and Inhibitory output of the model with imbalance α'.

12 Question 11: Biological Implications

The 4 models explored in this report allows us to understand that a symmetric ring structure is a very good model to retain the orientation selectivity nature of V1 compared with completely random connectivity. Some study does propose a random connectivity of the neuron as many animals such as rodents, lacks a functional map [2].

Even though, the symmetric ring structure is powerful, without inhibitory connections, the model could not scale to higher response rates stably. Therefore, a popular model of the V1 has been models which are somewhat similar to model 4 withe a balance ring structure.

As neurons in the V1 also posses contrast invariance, it is therefore reasonable to test for contrast invariance of our 4 baseline models. First we will scale our input vector by a contrast term which makes our feed forward input a function of both contrast and orientation. We used a method similar to that of [4] which utilised the Singular Value Decomposition of the tuning curves. By doing so, for all 4 models, the first singular value could explain more than 99% of the tuning curve. We therefore conclude that all models are contrast invariance. A 2D tuning curve from model 4 is shown in figure 9.

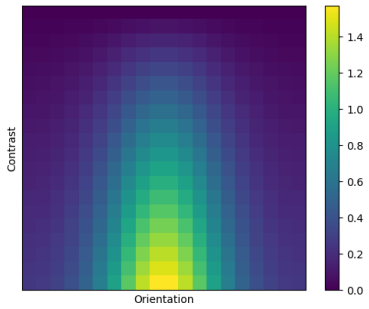

Figure 9: Tuning curve of model 4.

It is worth mentioning that we have modelled our V1 as a linear model which even though is easy to analyse, is unlikely to be the case. A better model would be for example the Stabilized Supralinear Network used in [4].

13 Conclusion

In this report, we have investigated 4 models of the recurrent network in the V1. We have found that a balance symmetric ring structure is the best at retaining stability and decoding accuracy.

We have also explored the importance of the input and output weights of the neurons in V1 in the role of decoding of the stimulus orientation.

Lastly, we argue that our all our model is also possesses contrast invariance and that the best V1 model biologically is the balance symmetric ring structure model. The classes used to simulate each model is attached as an appendix.

References

[1] Hubel DH. and Wiesel TN. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *J Physiol*, 160(1):106–54, January 1962.

[2] David Hansel and Carl van Vreeswijk. The mechanism of orientation selectivity in primary visual cortex without a functional map. *The Journal of Neuroscience*, 32(12):4049–4064, March 2012.

[3] Dylan R. Muir et al. Specific excitatory connectivity for feature integration in mouse primary visual cortex. *PLOS Computational Biology*, 13(12), December 2017.

[4] Nataliya Kraynyukova et al. The mechanism of orientation selectivity in primary visual cortex without a functional map. *Proceedings of the National Academy of Sciences*, 119(41), August 2022.

[5] Peter Dayan and L. F. Abbott. *Theoritical Neuroscience : Computational and Mathematical Modeling of Neural Systems*. The MIT Press, 2001.

A Simulation Code

```
import numpy as np
from math import atan2

class V1Connection:
    def __init__(self, m=200, n=200, kappa=np.pi/4, alpha=0.9):
        self.m = m
        self.n = n
        self.kappa = kappa
        self.alpha = alpha
        self.pref = np.linspace(0, 2*np.pi, m, endpoint=False)

    @staticmethod
    def spectral_abisca(A):
        if A.ndim != 2 or A.shape[0] != A.shape[1]:
            raise ValueError("Need a square matrix")
        return np.max(np.real(np.linalg.eigvals(A)))

    def scale_by_spectral_abisca(self, A, alpha=None):
        if alpha is None:
            alpha = self.alpha
        spec_ab = self.spectral_abisca(A)
        scale = spec_ab / alpha
        return A / scale

    def cric_gauss(self, x):
        return np.exp((np.cos(x) - 1) / (self.kappa))

    @staticmethod
    def pref_diff(pref_a, pref_b):
        return pref_b[None, :] - pref_a[:, None]

class NoRecurrenceWeights(V1Connection):
    def __init__(self, m=200, kappa=np.pi/4, alpha=0.9):
        n = m
        super().__init__(m, n, kappa, alpha)

    def get_weights(self):
        return np.zeros(shape=(self.m, self.n))

class RandomSymmetricConnectivityWeights(V1Connection):
    def __init__(self, m=200, kappa=np.pi/4, alpha=0.9):
        n = m
        super().__init__(m, n, kappa, alpha)

    def get_weights(self):
        W_tilde = np.random.randn(self.m, self.n)
        W = W_tilde + W_tilde.T
        return self.scale_by_spectral_abisca(W)

class SymmetricRingStructureWeights(V1Connection):
    def __init__(self, m=200, kappa=np.pi/4, alpha=0.9):
        n = m
        super().__init__(m, n, kappa, alpha)

    def get_weights(self):
        pref_matrix = self.pref_diff(self.pref, self.pref)
        W = self.cric_gauss(pref_matrix)
        return self.scale_by_spectral_abisca(W)

class BalanceRingStructureWeights(V1Connection):
    def __init__(self, m=200, kappa=np.pi/4, alpha=0.9):
```

```
        n = 2 * m
        super().__init__(m, n, kappa, alpha)

    def get_sub_weights(self):
        pref_matrix = self.pref_diff(self.pref, self.pref)
        W = self.cric_gauss(pref_matrix)
        return self.scale_by_spectral_abisca(W)

    def get_weights(self):
        EE = self.get_sub_weights()
        EI = - self.get_sub_weights()
        IE = self.get_sub_weights()
        II = - self.get_sub_weights()
        return np.concatenate((np.concatenate((EE, EI), axis=1),
                               np.concatenate((IE, II), axis=1)), axis=0)

class NetworkExecuter(V1Connection):
    def __init__(self, tau=0.02, W=None, B=np.eye(200), C=np.eye(200), sigma=1, kappa=np.pi/4,
                 alpha=0.9, delta_t=0.001) -> None:
        self.tau = tau
        self.W = W
        self.B = B
        self.C = C
        self.sigma = sigma
        self.delta_t = delta_t
        self.n = len(W)
        self.m = len(B[0])
        self.r_0 = np.zeros(self.n)
        super().__init__(self.m, self.n, kappa, alpha)

    def execute(self, t, orientation, W=None, plot=False):
        if W is not None and len(W) != self.n:
            raise ValueError("Invalid W dimension")
        elif W is not None:
            self.W = W
        rate = self.euler(t, orientation, plot=plot)
        output = self.C @ rate
        return output + self.sigma * np.random.randn(len(output))

    def get_feedforward_input(self, orientation, contrast=1):
        return contrast * self.cric_gauss(orientation - self.pref)

    def euler(self, t, orientation, plot=False, contrast=1):
        num_iterations = int(np.ceil(t / self.delta_t))
        rate = self.r_0
        h = self.get_feedforward_input(orientation, contrast=contrast)
        Bh = self.B @ h
        for _ in range(num_iterations):
            rate = rate + (self.delta_t / self.tau) * (-rate + self.W @ rate + Bh)
        return rate

def decoder(observations, pref):
    return atan2(np.sum(observations * np.sin(pref)), np.sum(observations * np.cos(pref)))

def decoding_error(predict, actual):
    return np.arccos(np.cos(predict - actual))
```