SciPy Sampa: Evolução da visão computacional utilizando redes neurais

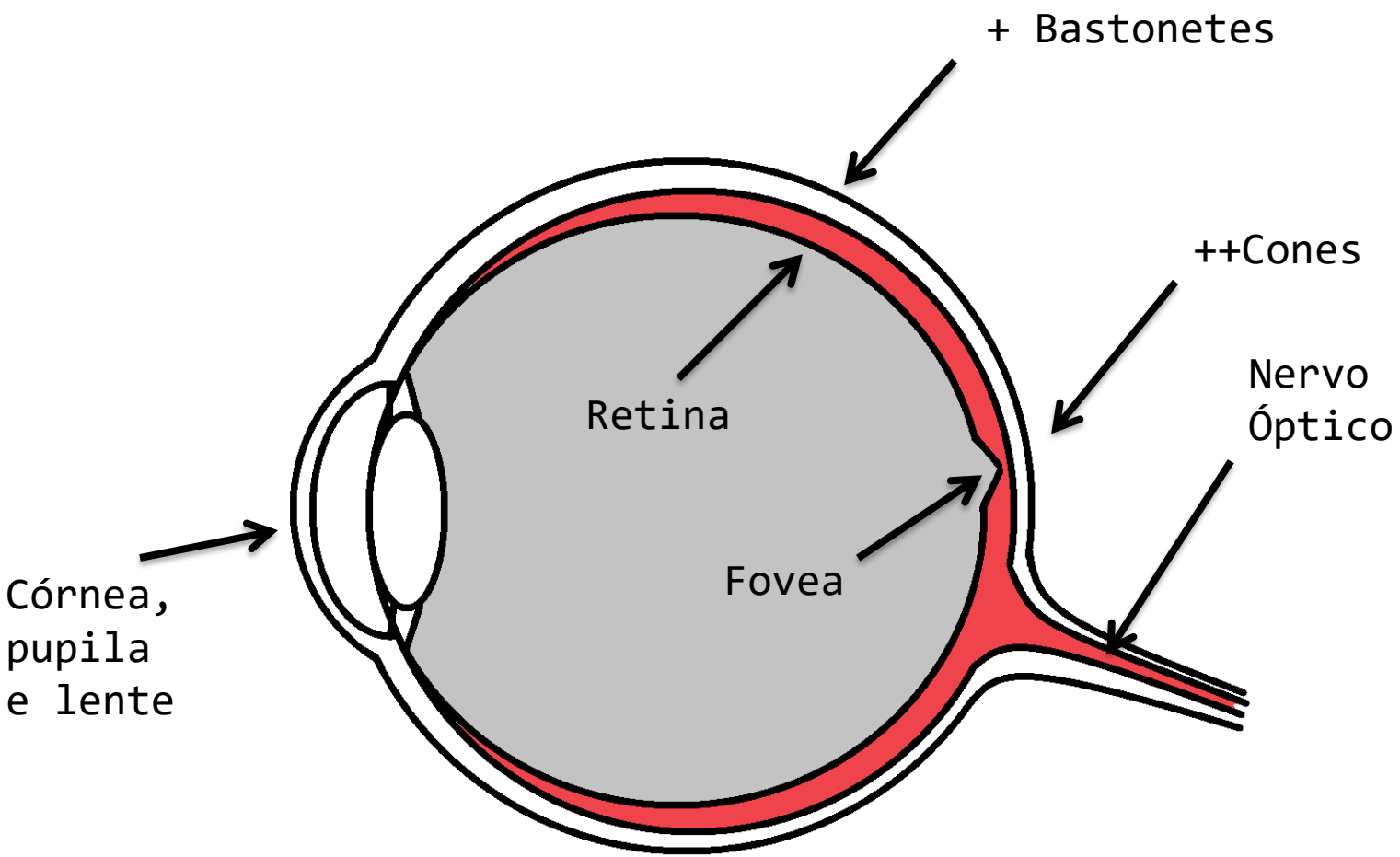
Potenciais de Utilização da Teoria BCM em Visão Computacional

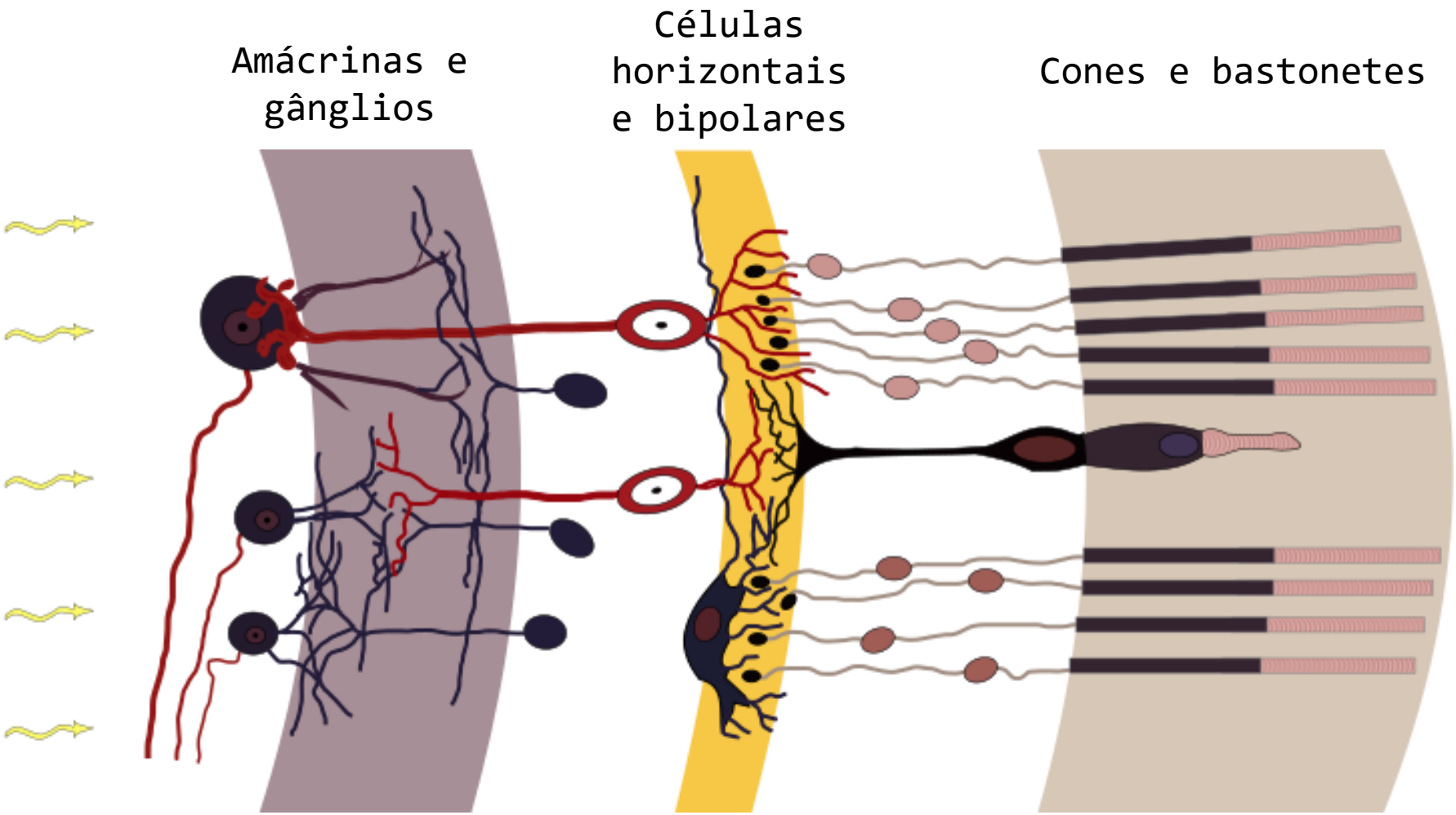
Por:
Jonas Santos Marma

jonas.marma@gmail.com
github.com/JonasMarma

14/03/2018

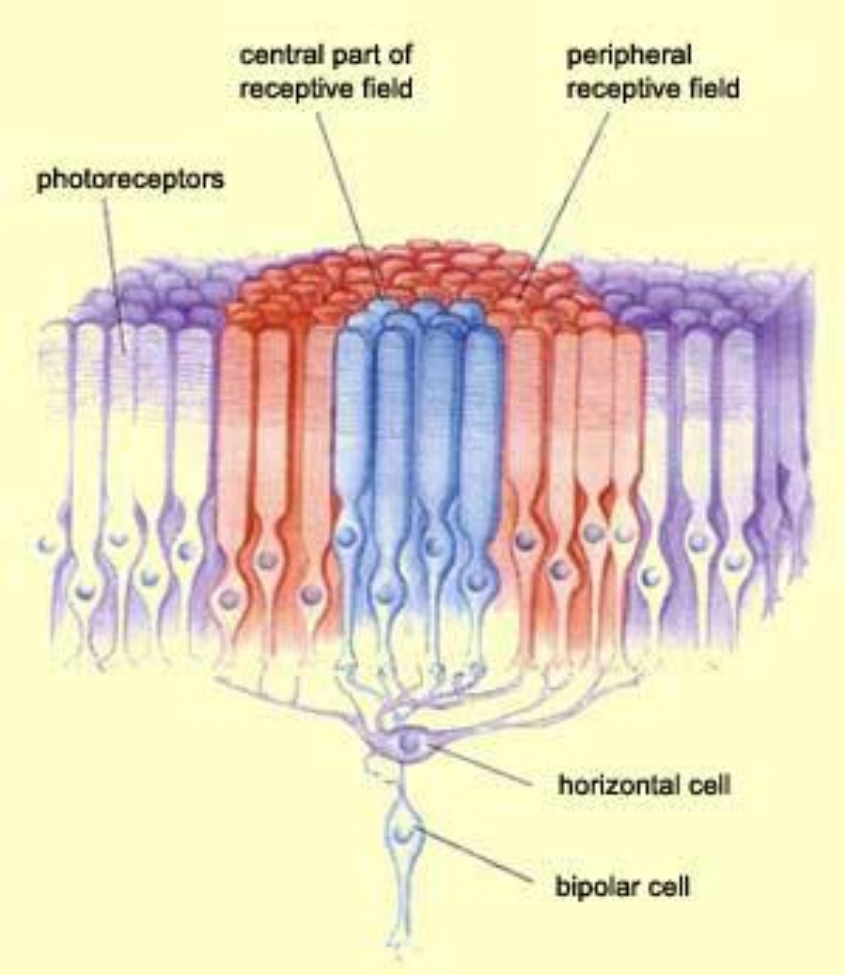
Introdução:
Olho humano





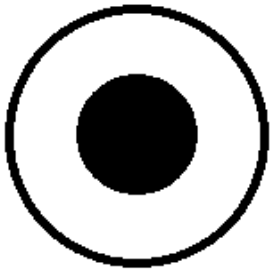
Axial organization of the retina (from Cajal, 1911). (Cajal, 1991): S. R. Y. CAJAL, Histologie Du Système Nerveux de lHomme et Des Vertébrés, Maloine, Paris, 1911

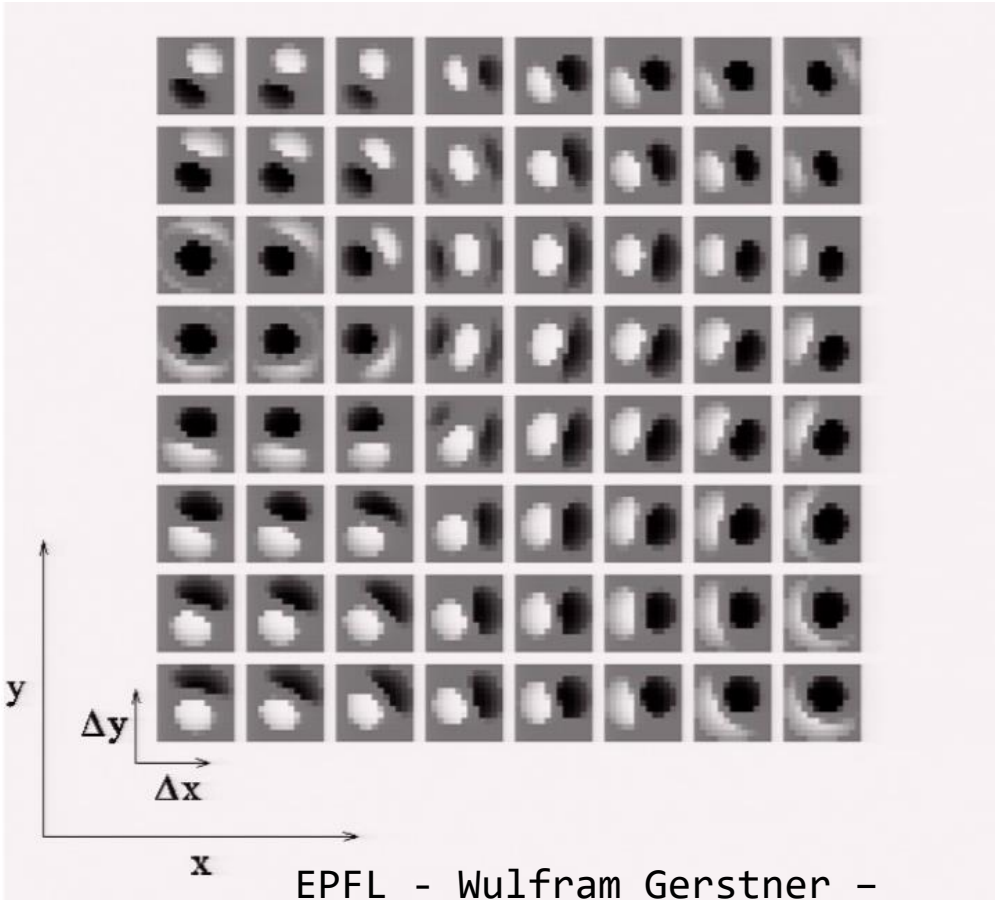
Campos receptivos



McGill University –
The Retina

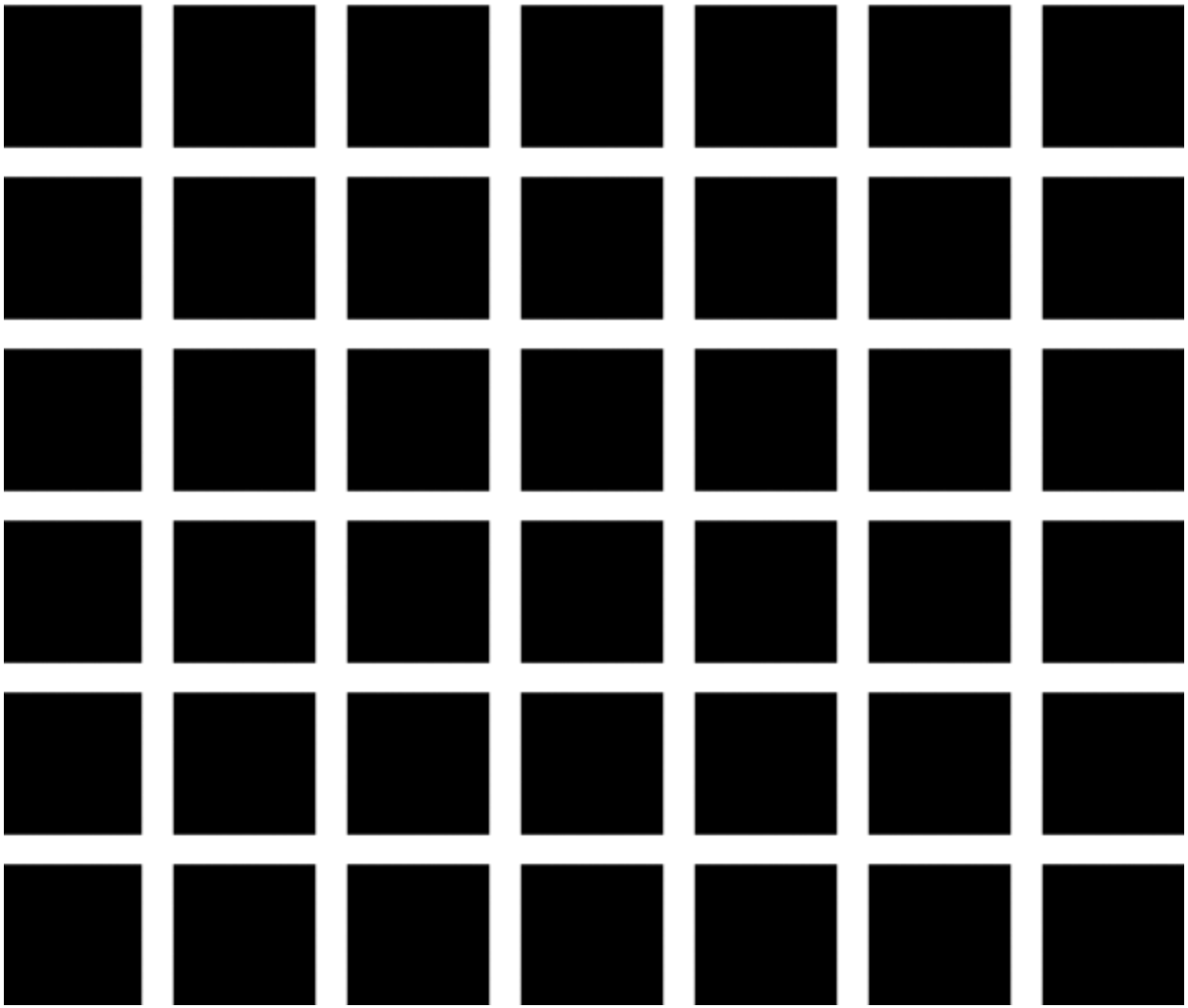
Desenvolvimento do campo receptivo (ESPECIALIZAÇÃO)

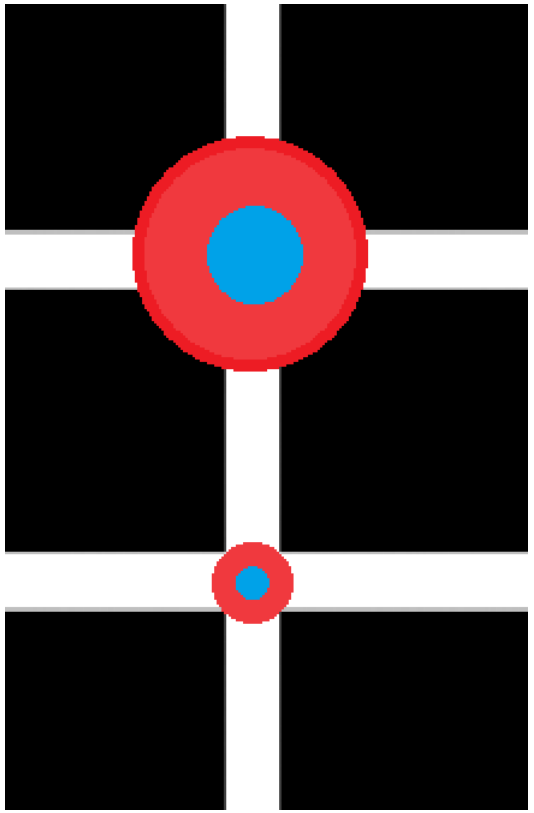




EPFL - Wulfram Gerstner – Biological Modeling of Neural Networks

Veja na prática!





Mas como?

Donald Hebb – 1949

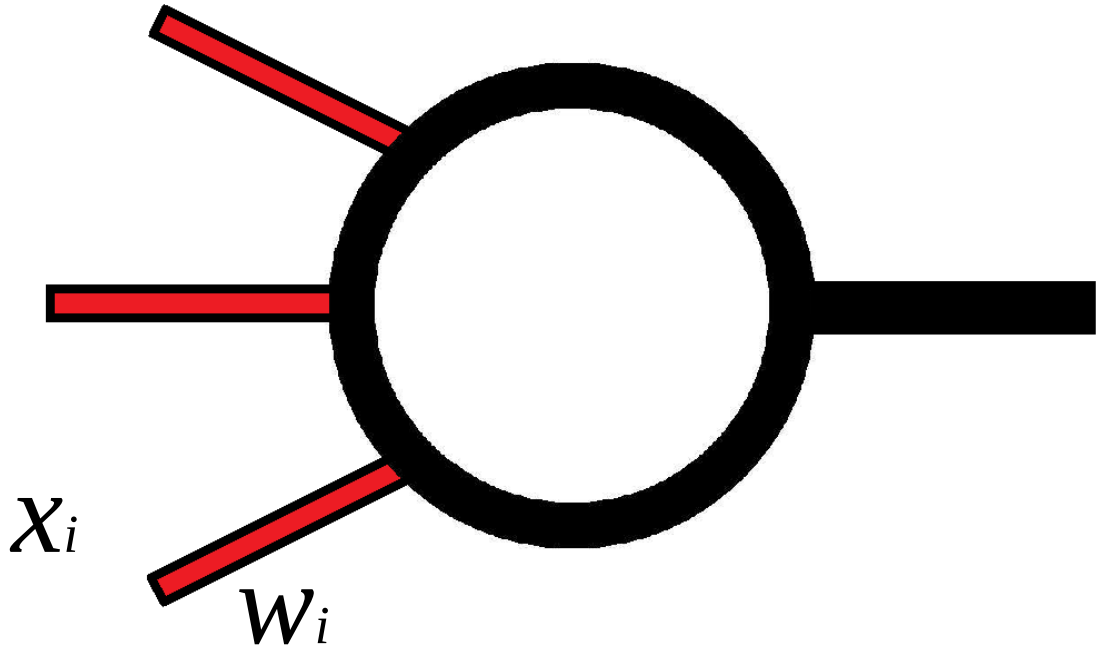
> *“Células que disparam juntas, permanecem conectadas.”*

Teoria BCM – 1982

Modelo hebbiano de aprendizado desenvolvido por Bienenstock, Cooper e Munro.

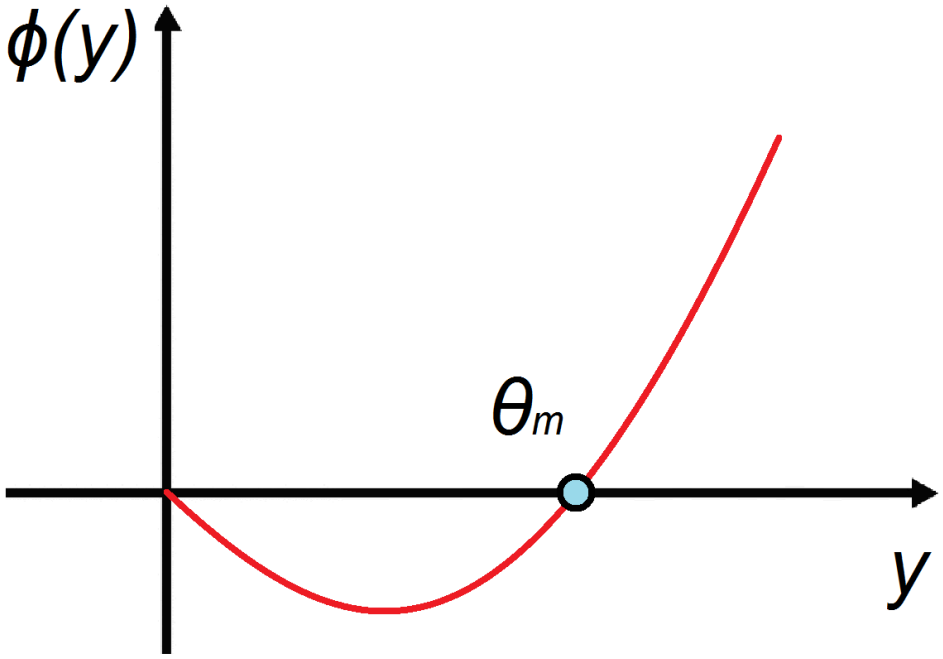
3 postulados:

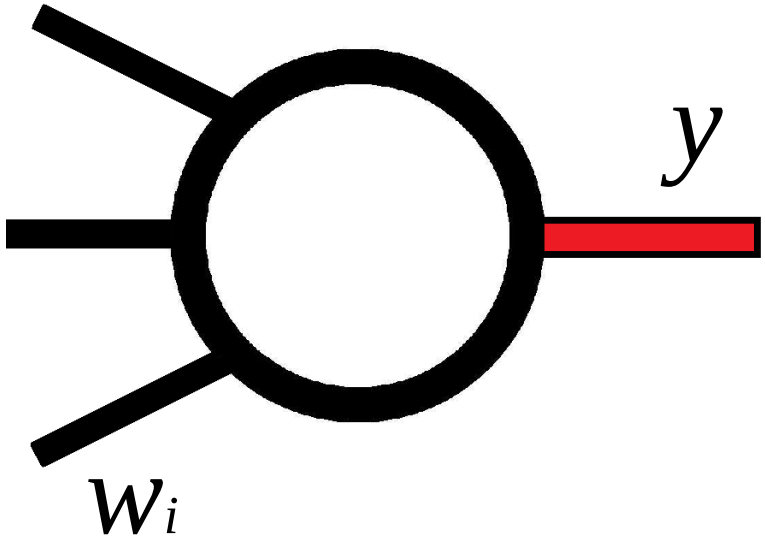
1º: Mudança nos pesos sinápticos $\propto$ Atividade pré-sináptica



3 postulados:

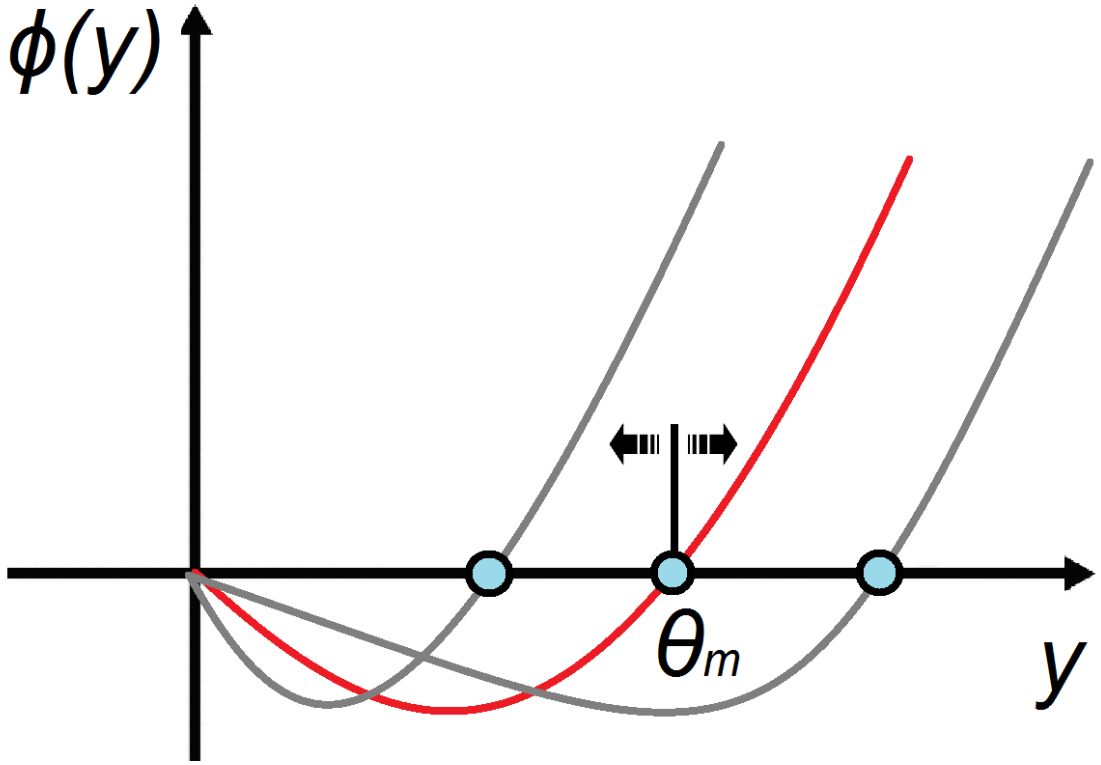
2º: Mudança nos pesos sinápticos $\propto$ Função da atividade pós-sináptica ϕ





3 postulados:

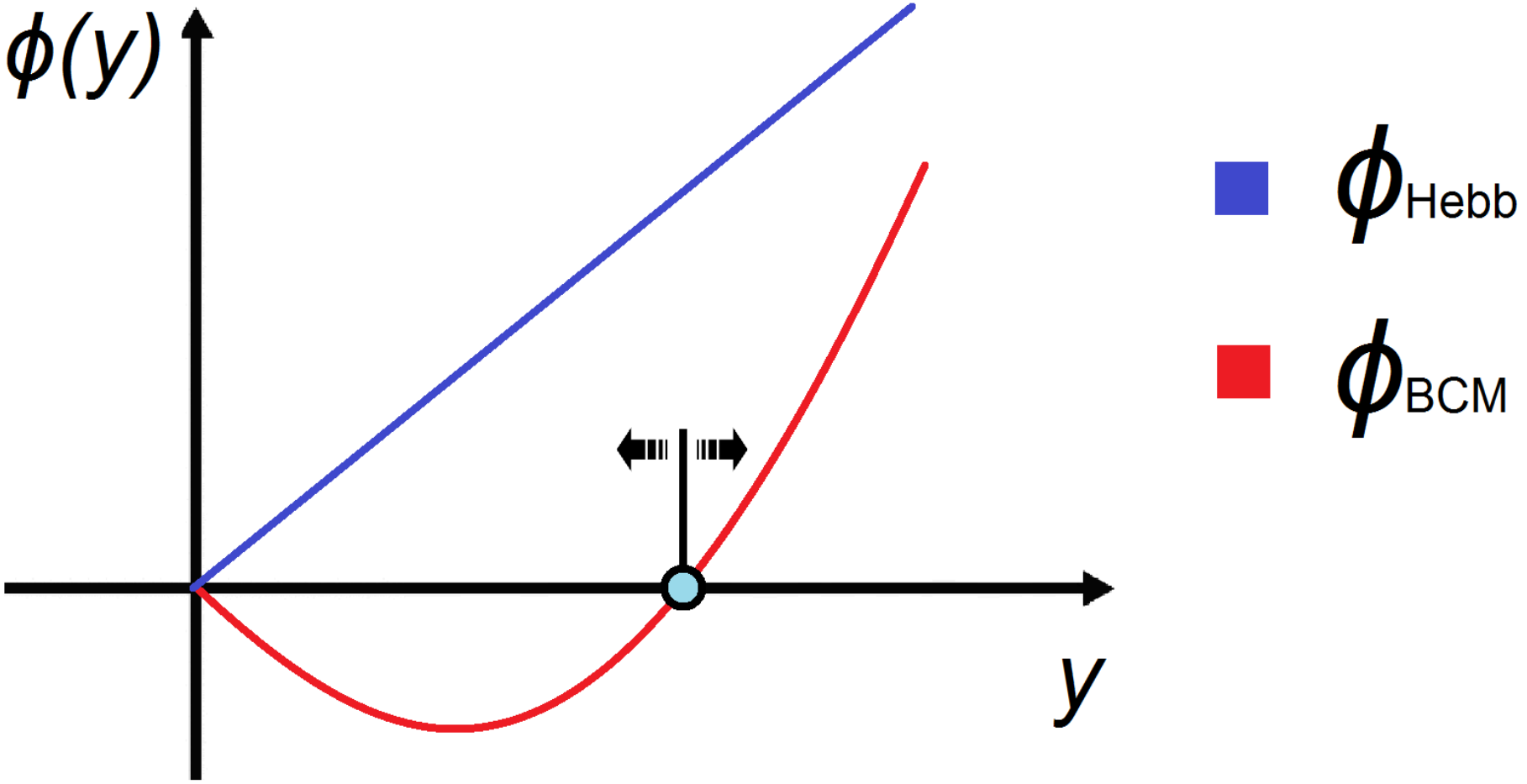
3º: O limiar θ_M depende do histórico de ativação y



Resumindo:

- $\frac{dw_i}{dt} \propto x_i$
- $\frac{dw_i}{dt} \propto \phi(y, \theta_M)$
- $\theta_{\mathrm{M}} = \mathrm{f}(\langle \mathrm{y} \rangle)$

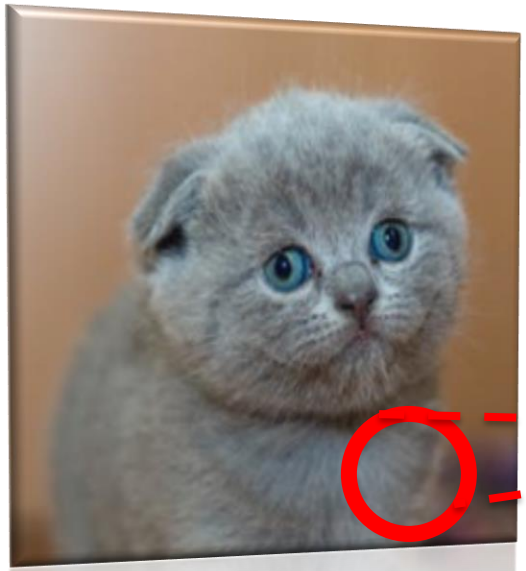
Hebb X BCM

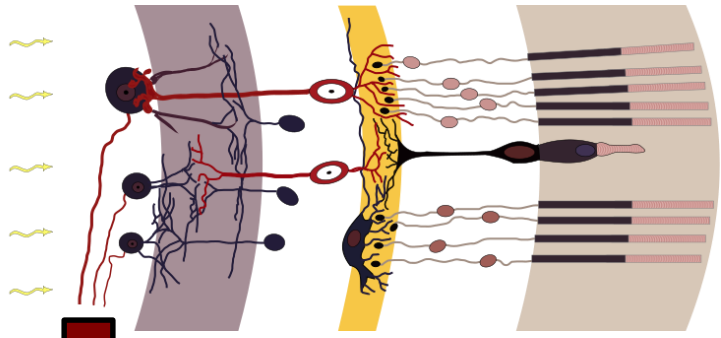


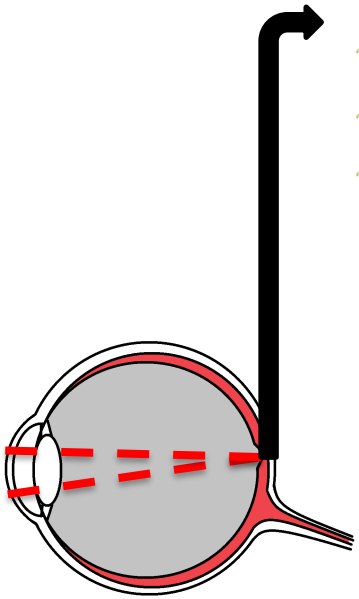
Formulação Law & Cooper - 1994

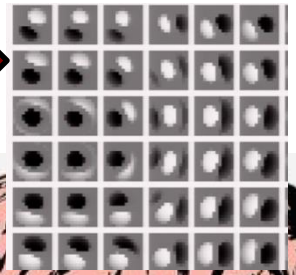
$$
\begin{aligned}
y &= \sigma\left(\sum_i w_i x_i\right) \\
\frac{dw_i}{dt} &= y(y-\theta_M)x_i/\theta_M \\
\theta_M &= E[y^2]
\end{aligned}
$$

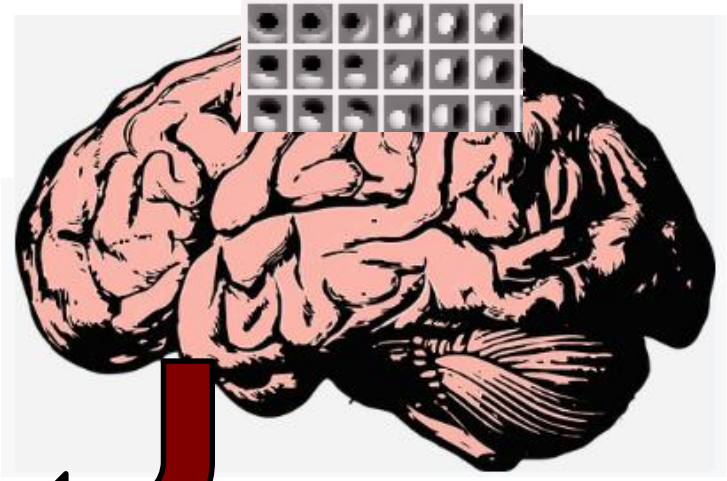
Visão “biológica”





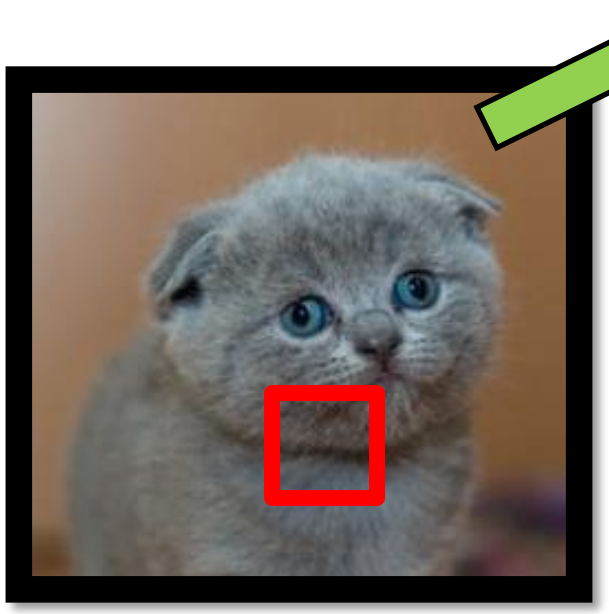






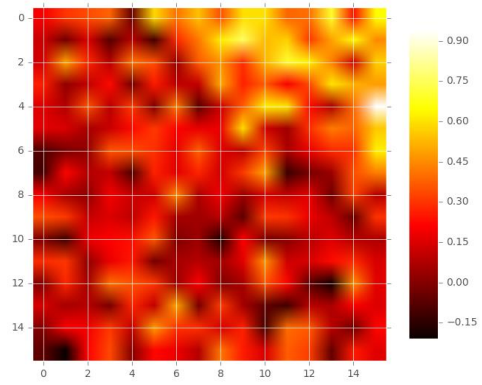
Gato!

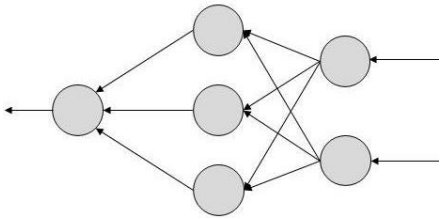
Visão computacional



153	213	200	150
45	145	144	45
230	144	150	155
215	211	49	90

BCM





Gato(94%)

```
def main(self):
    print("Digite a imagem que deseja processar:")
    img = raw_input()
    image = self.loadImage("images/" + img + ".bmp")
    normImage = self.normalize(image)
    inputVects = self.createInputs(normImage)

    iterations = 150000
    for i in range(0, iterations):
        #Pegando um quadrado aleatório
        randomPatch = np.random.randint(0, self.Npatches, 1)

        self.bcmTraining94(inputVects[randomPatch, :].flatten(), i)

    self.showWeights(self.weights)
```

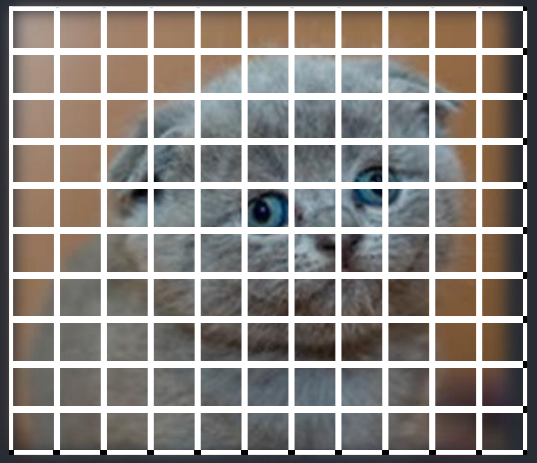
```python
def normalize(self, image):
    mean = np.mean(image)
    std = np.std(image)
    image -= mean
    image /= std
    return image
```

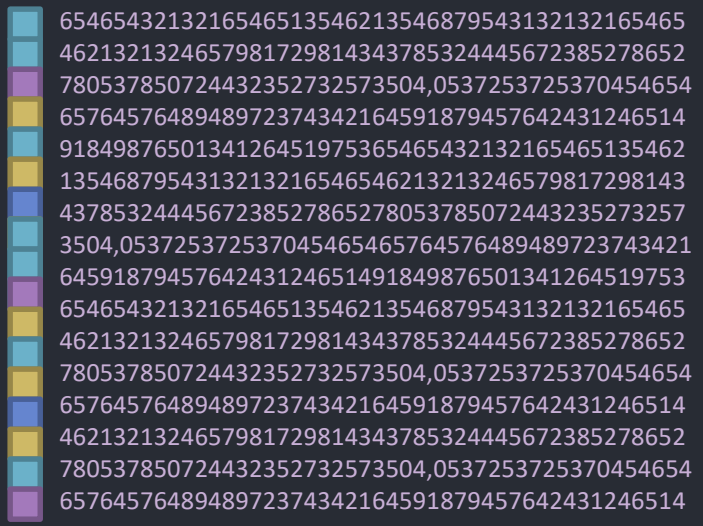
153	213	200	150
45	145	144	45
230	144	150	155
215	211	49	90

0.6	0.1	0.6	0.4
0.8	0.1	0.4	0.9
0.6	0.2	0.3	0.4
0.7	0.2	0.5	0.6

```
def main(self):
    print("Digite a imagem que deseja processar:")
    img = raw_input()
    image = self.loadImage("images/" + img + ".bmp")
    normImage = self.normalize(image)
    inputVects = self.createInputs(normImage)

    iterations = 150000
    for i in range(0, iterations):
        #Pegando um quadrado aleatório
        randomPatch = np.random.randint(0, self.Npatches, 1)

        self.bcmTraining94(inputVects[randomPatch, :].flatten(), i)

    self.showWeights(self.weights)
```

```python
def createInputs(self, image, xOverlay = 4, yOverlay = 4, patchSize = 16):
    patches = []
    self.patchSize = patchSize

    i = 0; j = 0
    while True:
        if i + patchSize > self.imageX:
            i = 0
            j += patchSize - yOverlay
        if j + patchSize > self.imageY:
            break

        #"Extrair" o quadrado
        patch = image[i:(i+patchSize) , j:(j+patchSize)].flatten()

        #preencher uma linha com o quadrado separando positivos e negativos:
        #1ª metade da linha
        patchHead = (patch > 0) * patch
        #2ª metade da linha
        patchTail = (patch < 0) * np.abs(patch)

        line = np.concatenate((patchHead, patchTail), axis = 0)

        patches.append(line)

        i += (patchSize - xOverlay)

    #Armazenar a quantidade de quadrados extraídos
    self.Npatches = len(patches)

    patches = np.array(patches)

    return patches
```




65465432132165465135462135468795431321321654654
46213213246579817298143437853244456723852786521
78053785072443235273250 4,0537253725370454654
65764576489489723743421645918794576424312465 14
91849876501341264519753654654321321654651354 62
13546879543132132165465462132132465798172981 43
43785324445672385278652780537850724432352732 57
3504,05372537253704546546576457648948972374342 1
64591879457642431246514918498765013412645197 53
65465432132165465135462135468795431321321654654
46213213246579817298143437853244456723852786521
78053785072443235273250 4,0537253725370454654
65764576489489723743421645918794576424312465 14
46213213246579817298143437853244456723852786521
78053785072443235273250 4,0537253725370454654
65764576489489723743421645918794576424312465 14
```
```

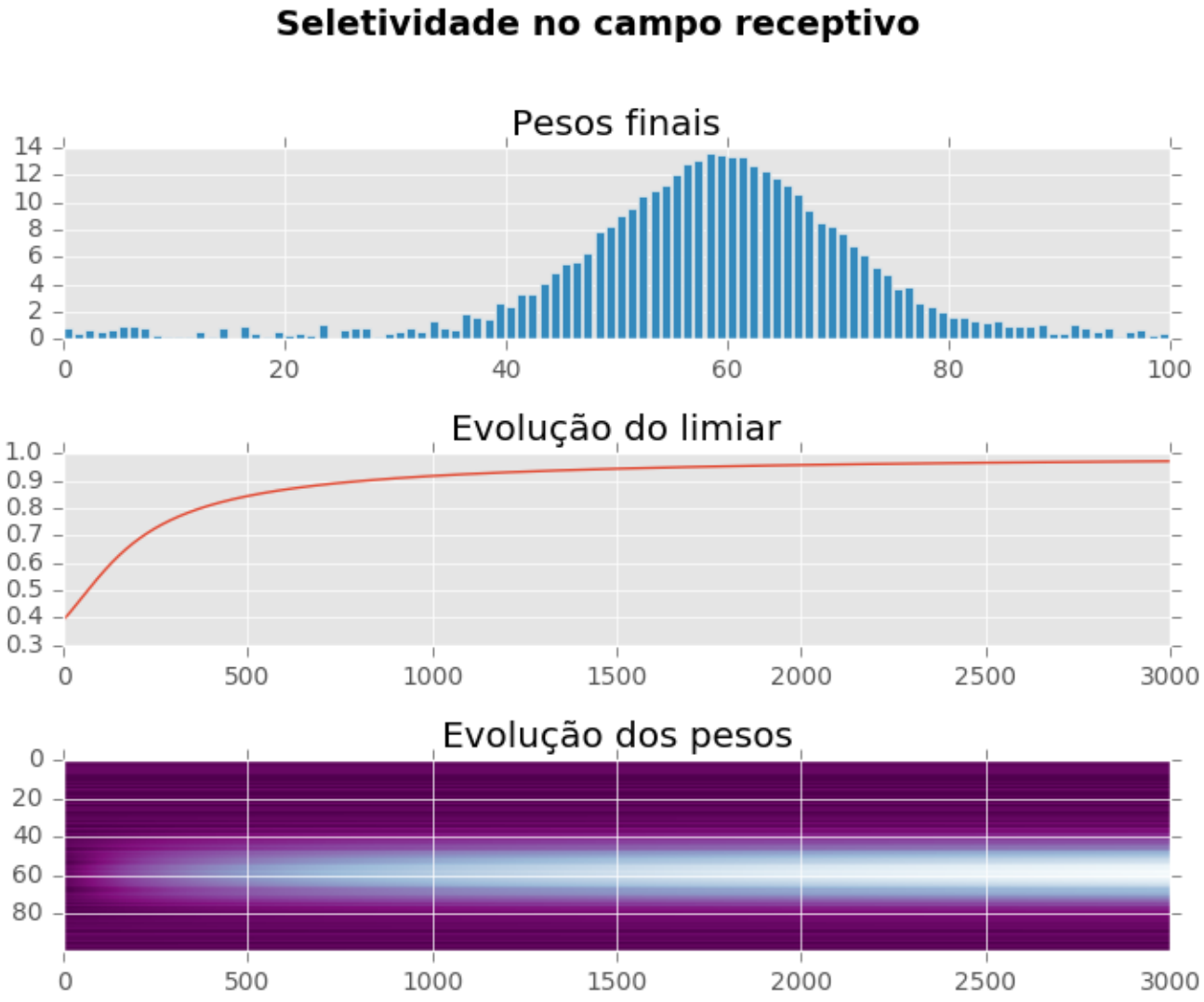
```
def main(self):
    print("Digite a imagem que deseja processar:")
    img = raw_input()
    image = self.loadImage("images/" + img + ".bmp")
    normImage = self.normalize(image)
    inputVects = self.createInputs(normImage)

    iterations = 150000
    for i in range(0, iterations):
        #Pegando um quadrado aleatório
        randomPatch = np.random.randint(0, self.Npatches, 1)

        self.bcmTraining94(inputVects[randomPatch, :].flatten(), i)

    self.showWeights(self.weights)
```

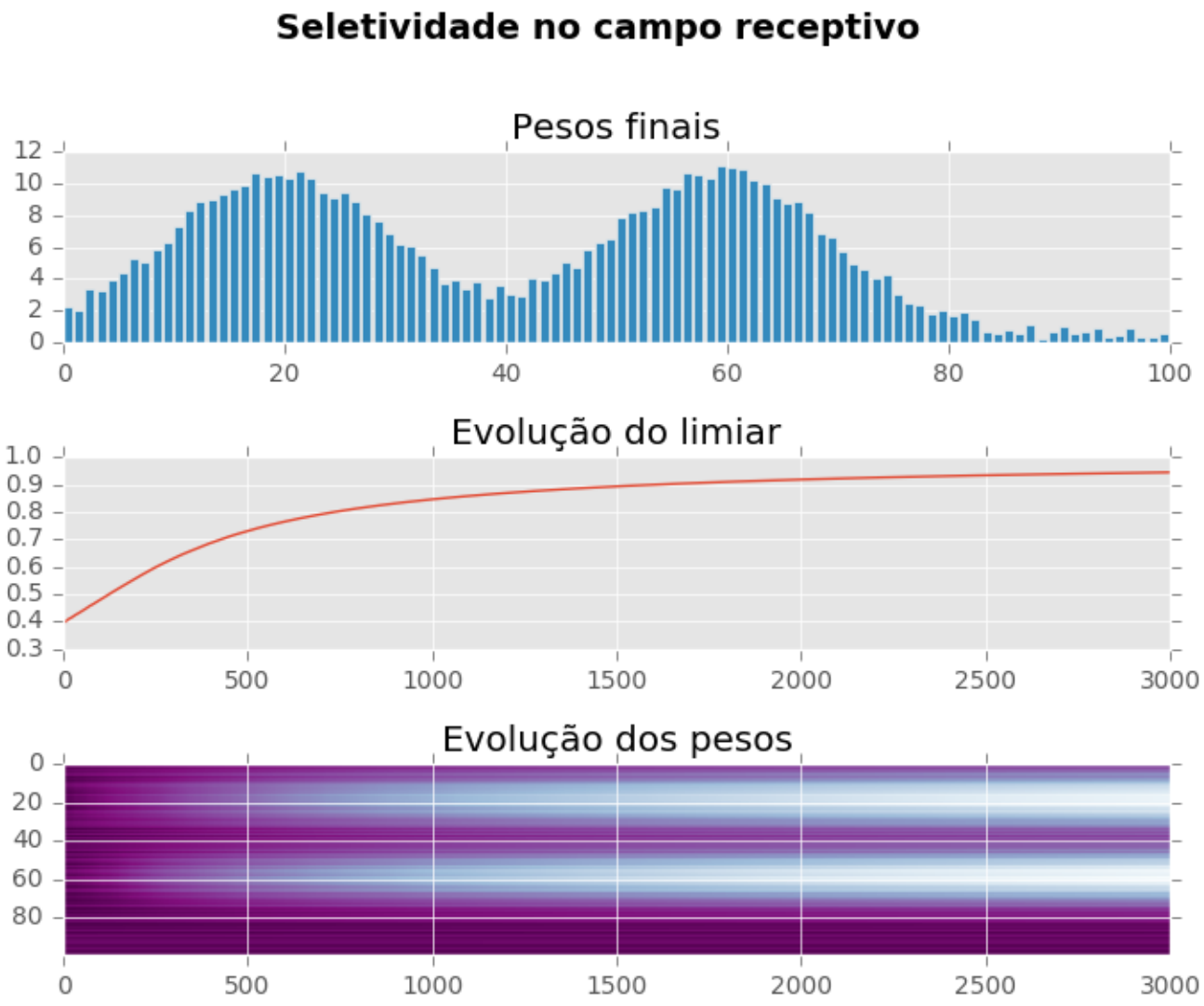
```python
def bcmTraining94(self, inputVect, iteration):
    #Calcular a resposta
    y = self.sigmoid(np.inner(self.weights, inputVect))

    #Alterar os pesos
    d_weights = (inputVect * (y ** 2 - y * self.theta))/self.theta
    self.weights += d_weights
    self.weights[self.weights < 0] = 0

    #Alterar o theta
    self.yIntegral += y
    self.theta = (self.yIntegral/(iteration+1))**2
```

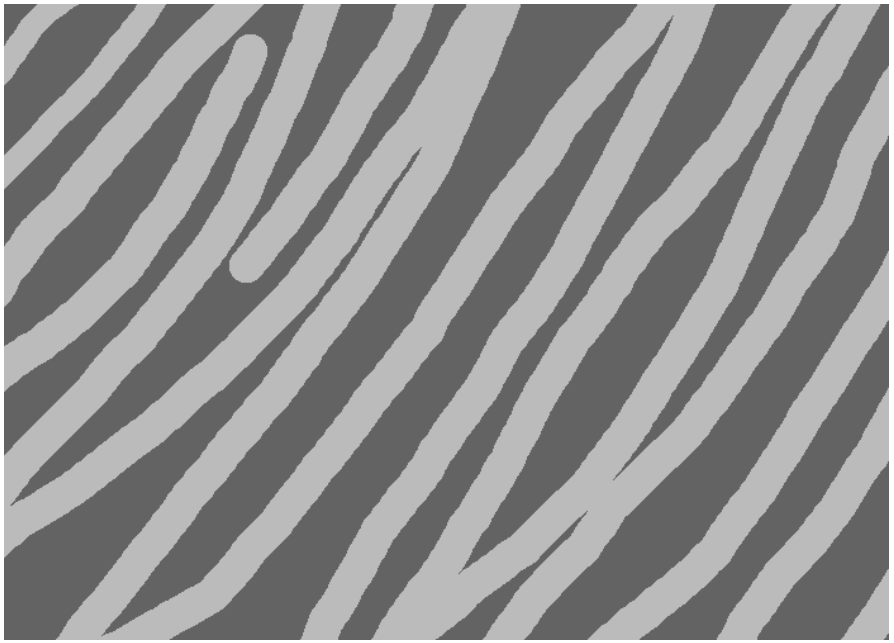
$$y = \sigma\left(\sum_i w_i x_i\right)$$

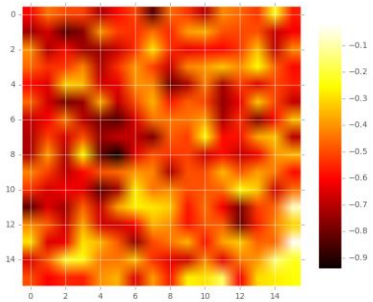
$$\frac{dw_i}{dt} = y(y - \theta_M)x_i/\theta_M$$

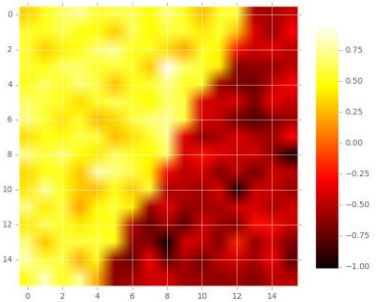
$$\theta_M = E[y^2]$$

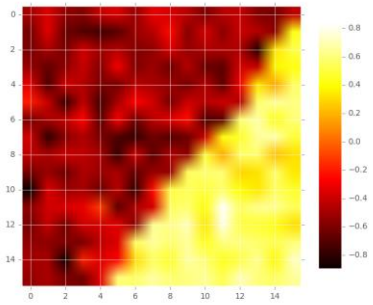

Seletividade no campo receptivo
Pesos finais
14
12
10
8
6
4
2
0
0
20
40
60
80
100
Evolução do limiar
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
500
1000
1500
2000
2500
3000
Evolução dos pesos
0
20
40
60
80
0
500
1000
1500
2000
2500
3000

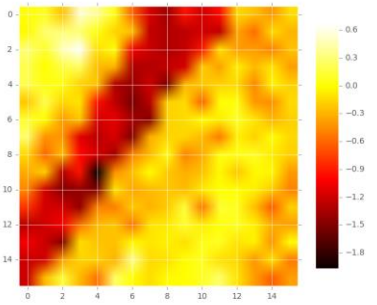

Seletividade no campo receptivo
Pesos finais
12
10
8
6
4
2
0
0
20
40
60
80
100
Evolução do limiar
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
500
1000
1500
2000
2500
3000
Evolução dos pesos
0
20
40
60
80
0
500
1000
1500
2000
2500
3000

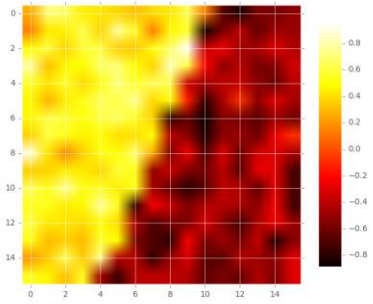


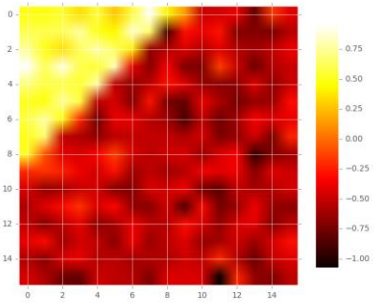


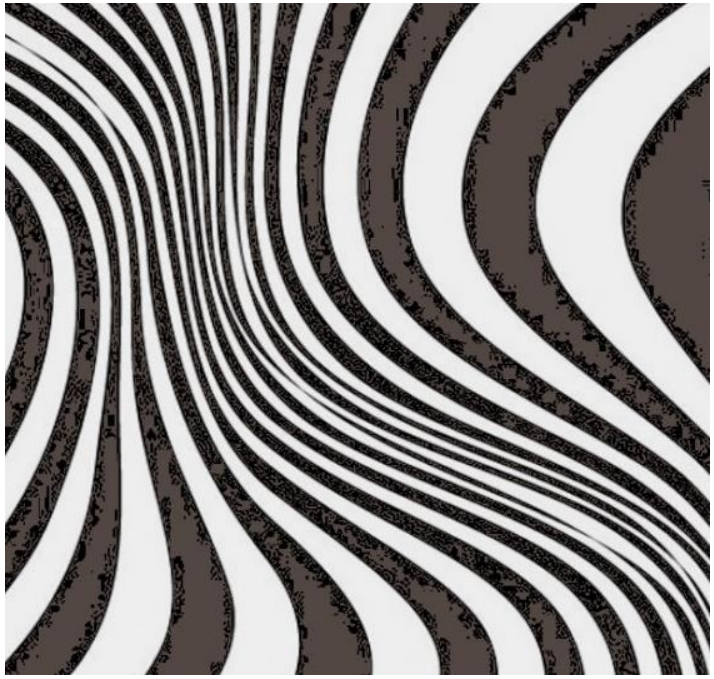


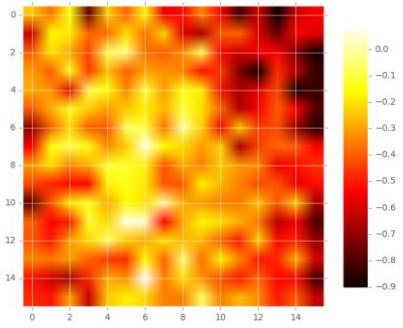


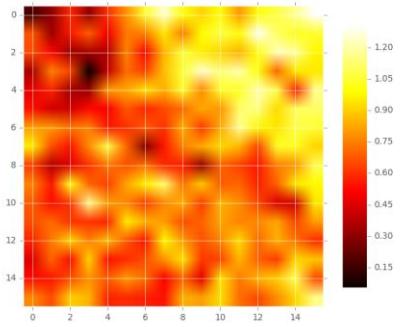


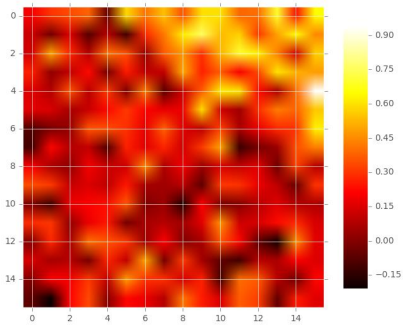


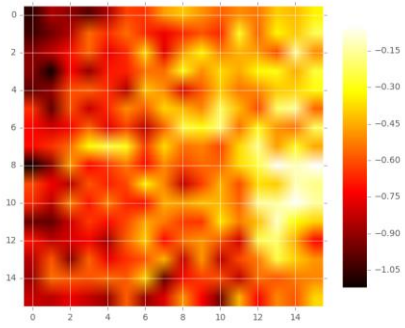


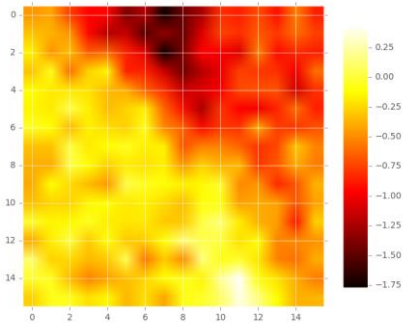


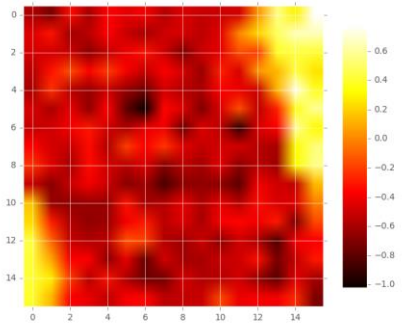

0 2 4 6 8 10 12 14
0.0 −0.1 −0.2 −0.3 −0.4 −0.5 −0.6 −0.7 −0.8 −0.9

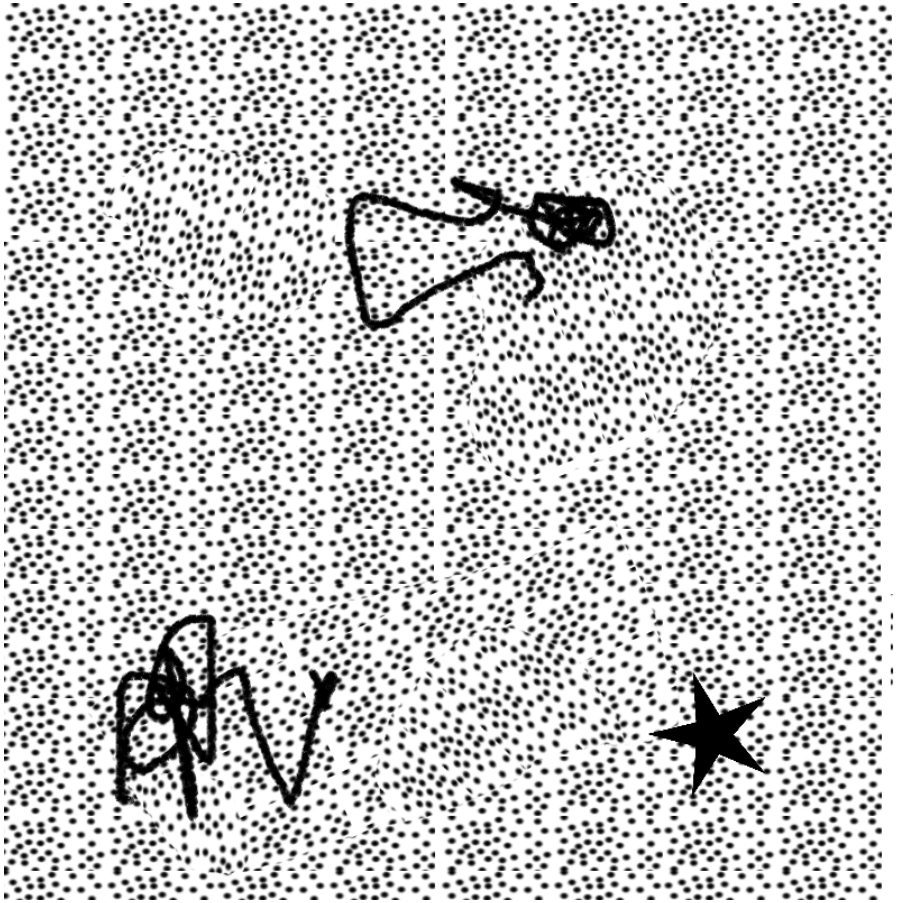

0 2 4 6 8 10 12 14
1.20 1.05 0.90 0.75 0.60 0.45 0.30 0.15

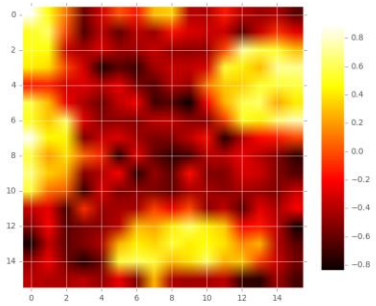

0 2 4 6 8 10 12 14
0.90 0.75 0.60 0.45 0.30 0.15 0.00 −0.15

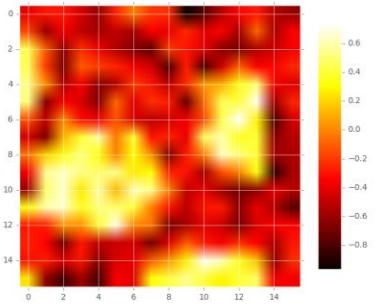

0 2 4 6 8 10 12 14
−0.15 −0.30 −0.45 −0.60 −0.75 −0.90 −1.05

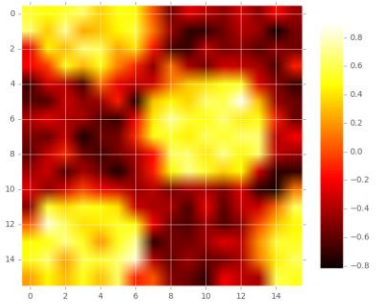

0 2 4 6 8 10 12 14
0.25 0.00 −0.25 −0.50 −0.75 −1.00 −1.25 −1.50 −1.75

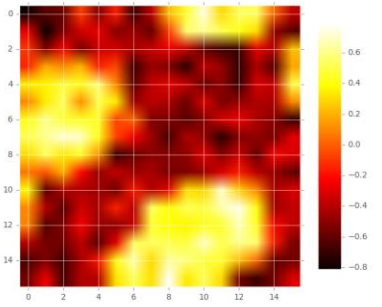

0 2 4 6 8 10 12 14
0.6 0.4 0.2 0.0 −0.2 −0.4 −0.6 −0.8 −1.0

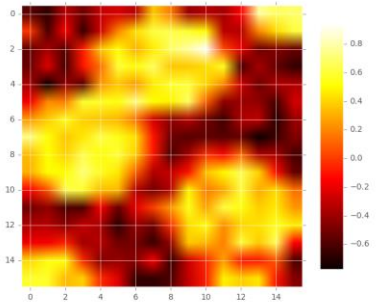


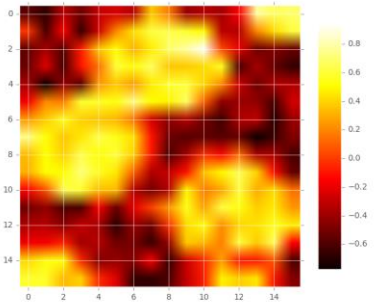


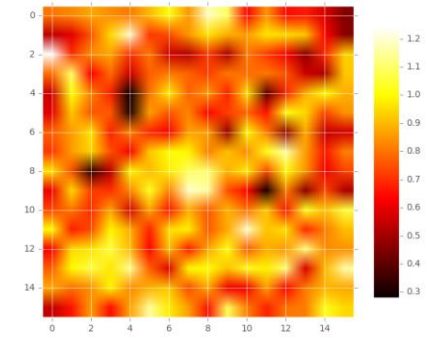


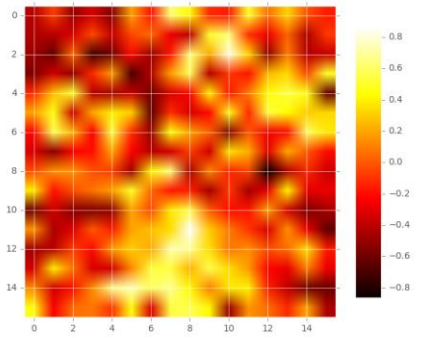


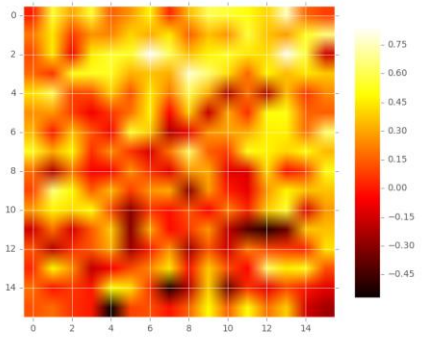


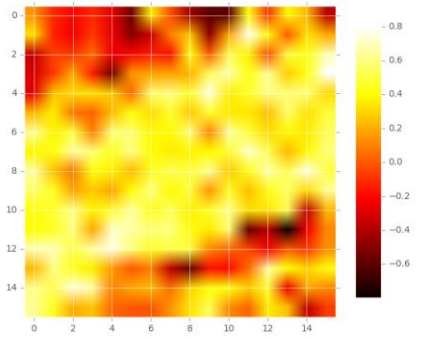


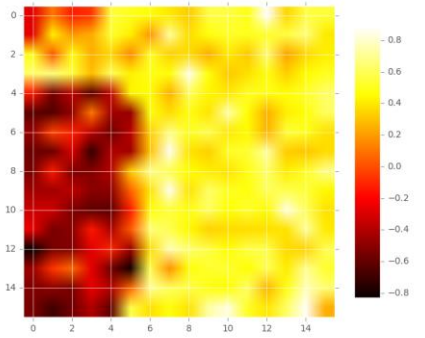


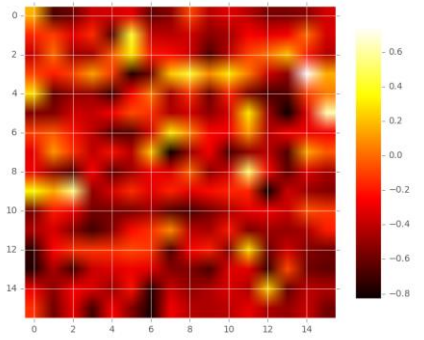

0 2 4 6 8 10 12 14
1.2 1.1 1.0 0.9 0.8 0.7 0.6 0.5 0.4 0.3


0 2 4 6 8 10 12 14
0.8 0.6 0.4 0.2 0.0 −0.2 −0.4 −0.6 −0.8


0 2 4 6 8 10 12 14
0.75 0.60 0.45 0.30 0.15 0.00 −0.15 −0.30 −0.45


0 2 4 6 8 10 12 14
0.8 0.6 0.4 0.2 0.0 −0.2 −0.4 −0.6


0 2 4 6 8 10 12 14
0.8 0.6 0.4 0.2 0.0 −0.2 −0.4 −0.6 −0.8


0 2 4 6 8 10 12 14
0.6 0.4 0.2 0.0 −0.2 −0.4 −0.6 −0.8

Obrigado!

jonas.marma@gmail.com

github.com/JonasMarma