

Trabajo práctico 4: Aprendizaje no supervisado

IGNACIO LEMBO FERRARI¹

¹ignaciolembo@ib.edu.ar

31 de octubre del 2023.

1. RED DE UNA CAPA LINEAL: IMPLEMENTACIÓN DE LA REGLA DE OJA

Se utilizó una red de una capa lineal con 4 entradas y una salida. La ecuación de salida es

$$V = \sum_{j=1}^4 w_j \xi_j. \quad (1)$$

La distribución de probabilidad de las entradas es una distribución gaussiana

$$P(\vec{\zeta}) = \frac{1}{(2\pi)^2 \sqrt{\Sigma}} \exp\left(-\frac{1}{2} \vec{\zeta}^T \Sigma^{-1} \vec{\zeta}\right) \quad (2)$$

donde Σ es la matriz de correlación

$$\Sigma = \begin{pmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{pmatrix} \quad (3)$$

Partiendo de pesos iniciales pequeños dados por una distribución uniforme en el intervalo $[-0.1, 0.1]$, se entrenó una red con datos de entrada ξ generados de la siguiente manera: En cada época, se generaron vectores ξ cuyas componentes se calcularon a partir de distribuciones de probabilidad gaussianas independientes dada por la Ec. 2. Luego, los vectores ξ se calcularon como

$$\xi = \Sigma^{1/2} \zeta, \quad (4)$$

de modo tal que estos nuevos vectores no tienen correlación entre sus componentes. Para actualizar los pesos en cada época, se utilizó la regla de Oja dada por

$$\Delta w_j = \eta V (\xi_j - V w_j). \quad (5)$$

donde η es la tasa de aprendizaje.

En el aprendizaje dado por la regla de Oja el vector de pesos se alinea con el autovector correspondiente al máximo autovalor de la matriz de correlación Σ . En este caso los autovalores de la matriz Σ son $\lambda = 1$

con multiplicidad 3 y $\lambda = 5$. En particular el autovector correspondiente al máximo autovalor $\lambda = 5$ es $\vec{v}_1 = (0.5, 0.5, 0.5, 0.5)$. Vale destacar que el vector de pesos también se podría alinear con vector antiparalelo a \vec{v}_1 , osea $-\vec{v}_1 = (-0.5, -0.5, -0.5, -0.5)$. Se realizaron estudios para distintas tasas de aprendizaje. En la Fig. 1 se muestra la evolución de las componentes del vector de pesos $\vec{w} = (w_1, w_2, w_3, w_4)$ en función de las épocas. Se observa que las componentes de dicho vector convergen a las componentes del autovector \vec{v}_1 o $-\vec{v}_1$.

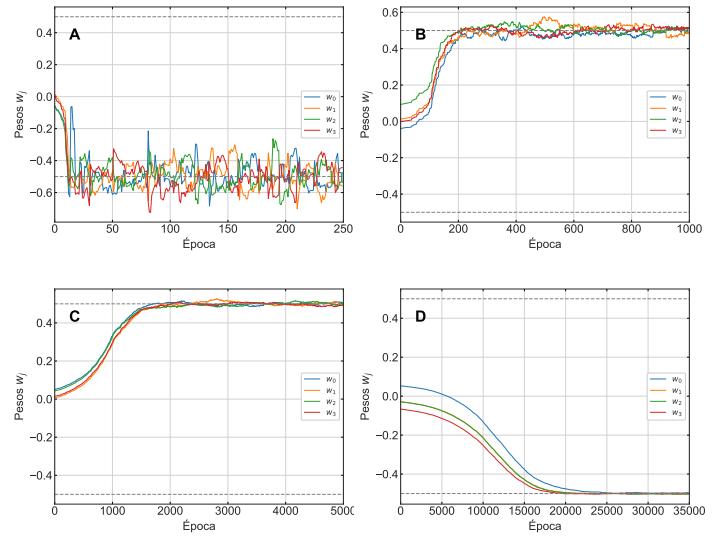


Fig. 1. Evolución de los pesos de la red neuronal para (A) $\eta = 0.01$, (B) $\eta = 0.001$, (C) $\eta = 0.0001$ y (D) $\eta = 0.00001$

En la Fig. 2 se muestra la evolución del producto escalar del vector de pesos con los autovectores de la matriz Σ en función de las épocas. Se observa que el producto escalar con el autovector \vec{v}_1 correspondiente al máximo autovalor de Σ converge a 1 o -1 mientras que los productos escalares con el resto de los autovectores corresponde a 0, indicando que \vec{w} es ortogonal a los mismos.

En todos los casos se observa que al aumentar la tasa de aprendizaje las curvas son menos ruidosas pe-

ro a la red neuronal le lleva más épocas converger a los valores mencionados. No obstante, siempre converge a los valores mencionados para todas las tasas de aprendizaje.

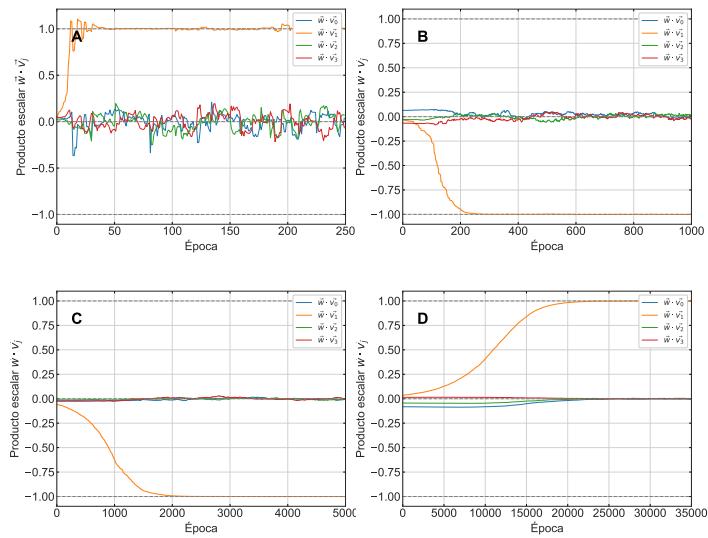


Fig. 2. Evolución del producto escalar del vector pesos con los autovectores de la matriz Σ para (A) $\eta = 0.01$, (B) $\eta = 0.001$, (C) $\eta = 0.0001$ y (D) $\eta = 0.00001$. El autovector \vec{v}_1 corresponde al máximo autovalor $\lambda = 5$ de la matriz Σ .

2. RED NEURONAL DE KOHONEN

La red neuronal de Kohonen es una red neuronal artificial no supervisada que se utiliza para el aprendizaje competitivo de mapas topológicos. Se implementó dicha red con dos neuronas de entrada y diez neuronas de salida.

En cada paso de aprendizaje se selecciona una neurona de salida i^* que minimiza la distancia euclídea $\|\vec{\xi} - \vec{w}_i\|$. Luego, se actualizan los pesos de la neurona seleccionada y de sus vecinas de acuerdo a la regla de aprendizaje de Kohonen dada por

$$\Delta w_{ij} = \eta \Lambda(i, i^*) (\xi_i - w_{ij}) \quad (6)$$

donde η es la tasa de aprendizaje y Λ es la función vecindad. En este trabajo se utilizó la función vecindad gaussiana dada por

$$\Lambda \propto \exp\left(-\frac{(i - i^*)^2}{2\sigma^2}\right). \quad (7)$$

La función vecindad se utiliza para determinar el grado de influencia de las neuronas vecinas en la actualización de los pesos de la neurona seleccionada. Es decir,

para valores grandes de σ la función vecindad es más ancha, por lo que las neuronas de salida se ven más afectadas por las neuronas vecinas. En cambio, para valores pequeños de σ la actualización de los pesos es más local y las neuronas no se afectan tanto entre sí.

A. Evolución temporal de la red

Para estudiar el efecto del tiempo de entrenamiento en la posición asintótica de los pesos sinápticos, se iniciaron los pesos de cada neurona de salida en coordenadas aleatorias en el rango $x \in [-0.2, 0.2]$ e $y \in [0, 0.2]$. Se alimentó las neuronas de entrada con una distribución

$$P(\vec{\xi}) = P(r, \theta) = \begin{cases} \text{constante si } r \in [0.9, 1.1], \theta \in [0, \pi] \\ 0 \text{ si no} \end{cases} \quad (8)$$

donde r y θ son las coordenadas polares del vector ξ . Se utilizó la función vecindad gaussiana

En la Fig. 3 se muestra la trayectoria de las neuronas de salida para distintas épocas. Se observa que, al evolucionar la red temporalmente, las neuronas convergen rápidamente en dirección radial hacia el anillo, pero luego se observa una evolución muy lenta en dirección tangencial. Luego de las 100000 épocas se observan cambios muy pequeños. Se puede ver, para valores grandes de épocas, como las neuronas se ordenan en el anillo cubriendo todo el espacio.

B. Efecto del parámetro σ en la evolución de la red

Se realizaron experimentos para distintos valores del parámetro σ de la función vecindad gaussiana. En la Fig. 4 se muestra la trayectoria de las neuronas de salida entrenadas con $\eta = 0.001$ y 100000 épocas para distintos valores de σ . Se observa que para valores grandes de σ la evolución de la red cubre más espacio y las neuronas se encuentran más superpuestas pero no se encuentran centradas en el anillo, si no, en el borde interior del mismo. Las neuronas siempre se ordenan. Para valores pequeños de σ , en particular para $\sigma = 0.5$, si bien hay un orden, no es tan claro como en el resto de los casos y se observa que sólo algunas neuronas llegan a evolucionar hasta el anillo, esto se debe a que la función vecindad es muy angosta, por lo que las neuronas de salida se ven menos afectadas por las neuronas vecinas. Luego, hay neuronas, cuya actualización de pesos prácticamente no se modifica al ser tan localizada la función Λ .

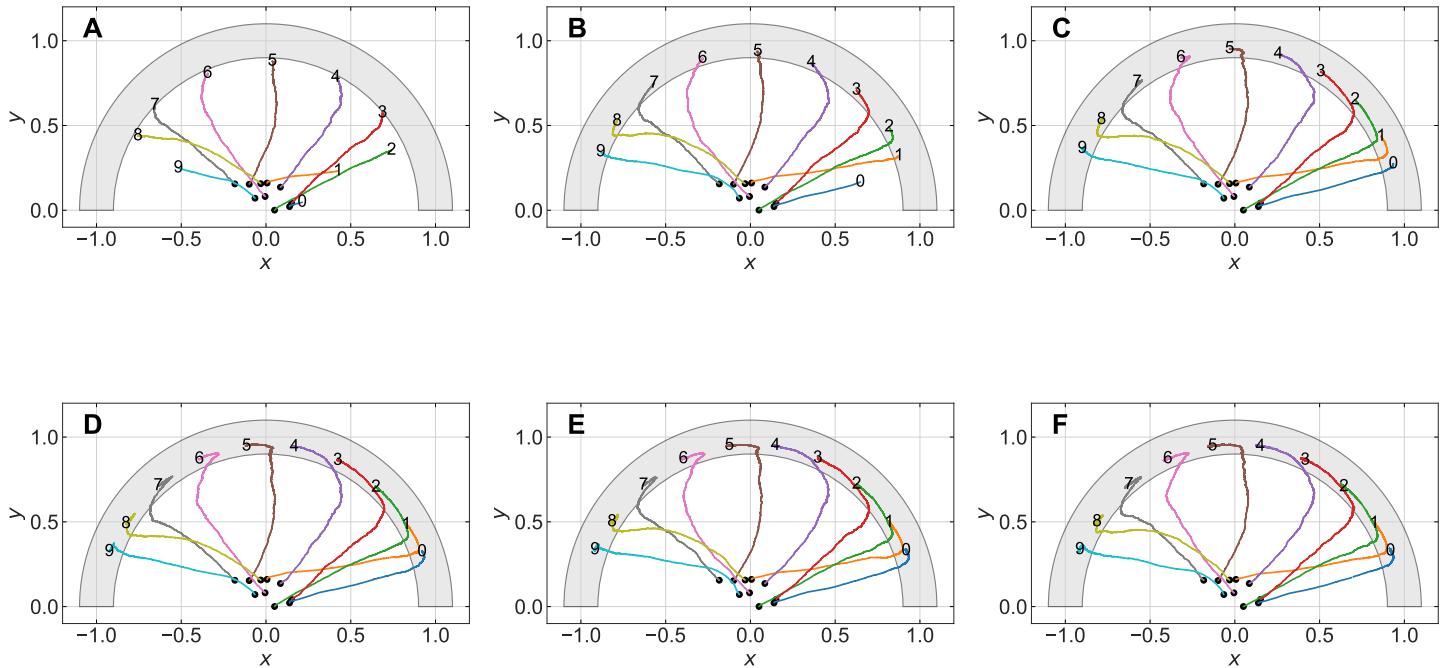


Fig. 3. Trayectoria de las neuronas de salida para una red de Kohonen con $\sigma = 1$, $\eta = 0.001$ y (A) 10000 épocas, (B) 25000 épocas, (C) 50000 épocas y (D) 100000 épocas, (E) 200000 épocas, (F) 500000 épocas.

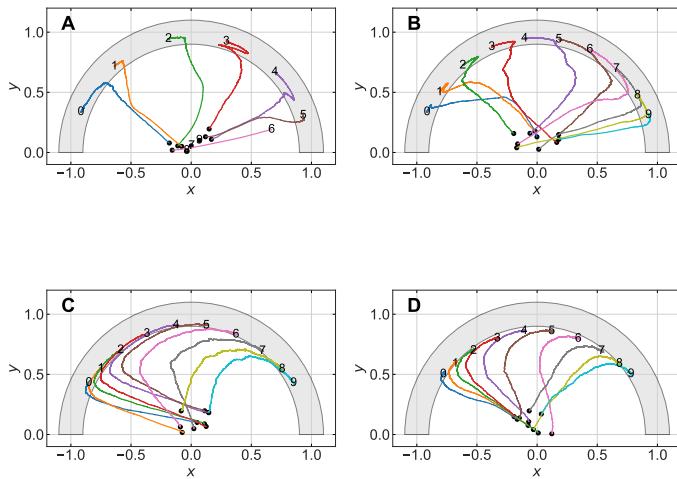


Fig. 4. Trayectoria de las neuronas de salida para una red de Kohonen con $\eta = 0.001$ para 100000 épocas y (A) $\sigma = 0.5$, (B) $\sigma = 1.0$, (C) $\sigma = 1.5$ y (D) $\sigma = 2.0$.

C. Parámetro σ adaptativo

Inspirado en la sección anterior (B) se implementó un parámetro σ adaptativo que disminuye con el tiempo de entrenamiento de la forma $\sigma(t) = \sigma_0/t$ y se

realizaron experimentos similares a los de dicha sección. Se alimentó la red con entradas dadas por una distribución constante dentro de una semicorona, un triángulo equilátero y un cuadrado donde la probabilidad de tener una entrada en la mitad superior del mismo es el doble que en la mitad inferior. En todos los casos, se entrenó a la red durante 100000 épocas y $\eta = 0.01$.

En el primer experimento, los pesos iniciales se tomaron en una región pequeña de modo que las neuronas empiezan todas cerca. En la Fig. 5 se observa que las neuronas inicialmente se mueven todas juntas y luego en un punto se separan, cubriendo todo la topología de las neuronas de entrada. Se detectó que, este punto, para todas las topologías estudiadas, resulta ser el centro de masa de la distribución de entradas. Luego, el centro de masa de esa topología es el punto hacia donde las neuronas viajan y en dicho punto se separan para cubrir todo el espacio. Esto parece ser independiente de donde inicen las neuronas.

En base a esto último, se realizaron experimentos donde las neuronas de salida comienzan dispersas en las cercanías de la región donde la distribución de las entradas es no nula. Los resultados se

muestran en la Fig. 6. En este caso tambien se ve que las neuronas convergen al centro de masa y luego se distribuyen cubriendo todo el espacio

Por ultimo, se observa que en general, las neuronas luego de pasar todas por el centro de masa, se acomodan de forma ordenada cubriendo el espacio. Este resultado parece mostrar que la red neuronal cuando $\sigma = \sigma_0/t$ encuentra el centro de masa de la distribución de entradas estudiadas, al menos para valores convenientes de los parámetros η , σ_0 y número de épocas. Sin sigma adaptativo no se han podido replicar estos resultados.

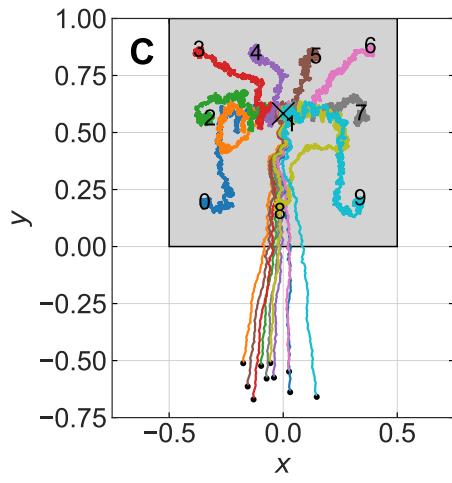
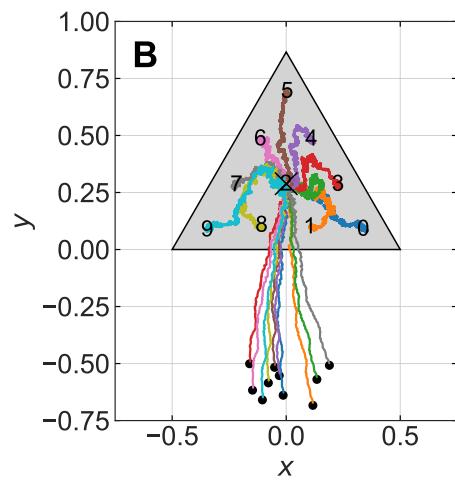
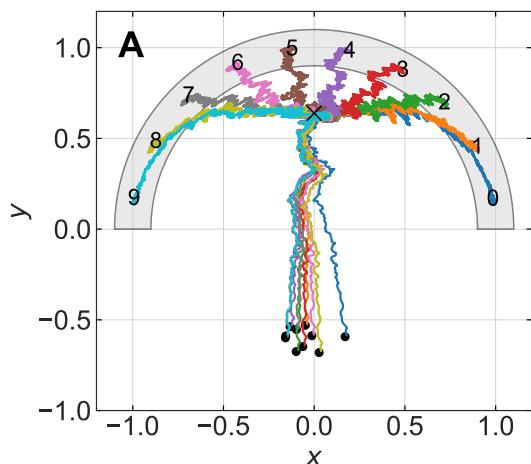


Fig. 5. Trayectoria de las neuronas de salida para una red de Kohonen con $\eta = 0.01$ para 100000 épocas para distintas topologías de la distribución de probabilidad uniforme de las neuronas de entrada (A) corona semicircular, (B) triángulo equilátero, (C) cuadrado con el doble de probabilidad de tener una entrada en la mitad superior que en la mitad inferior. Todas las neuronas empiezan distribuidas aleatoriamente y cerca entre sí. Se marca con \times el centro de masa de la distribución de entradas.

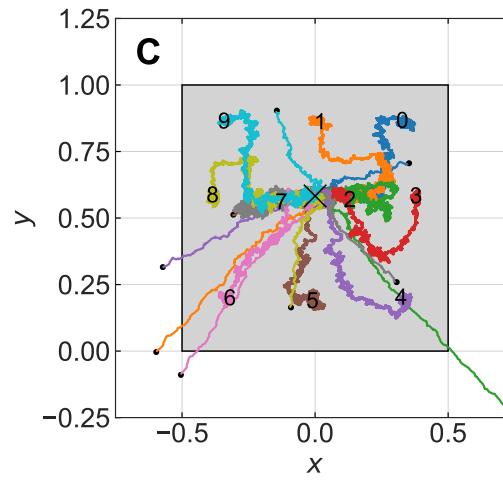
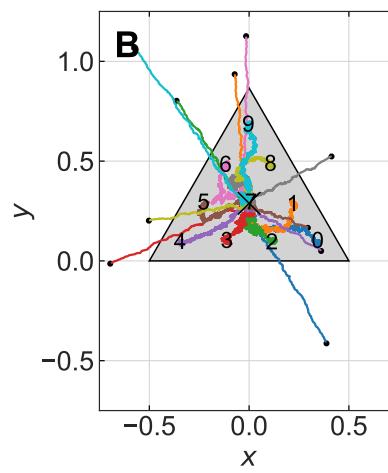
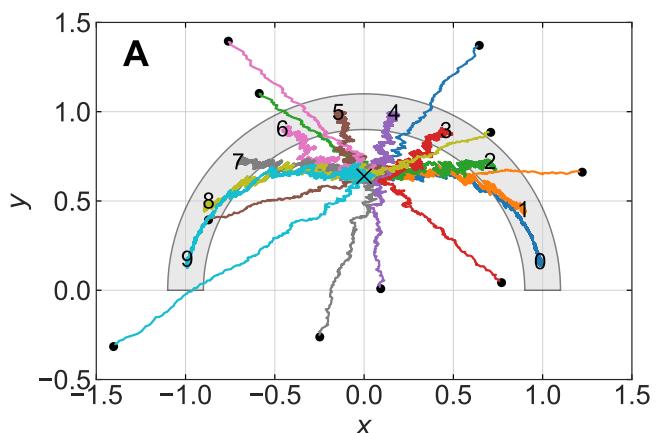


Fig. 6. Trayectoria de las neuronas de salida para una red de Kohonen con $\eta = 0.01$ para 100000 épocas para distintas topologías de la distribución de probabilidad uniforme de las neuronas de entrada (A) corona semicircular, (B) triángulo equilátero, (C) cuadrado con el doble de probabilidad de tener una entrada en la mitad superior que en la mitad inferior. Todas las neuronas empiezan en distribuidas aleatoriamente en las cercanías de la topología de entrada. Se marca con \times el centro de masa de la distribución de entradas.

A. APÉNDICE

A. Ejercicio 1

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from tqdm import tqdm
4
5 #Ploteo
6 import seaborn as sns
7 #sns.axes_style("whitegrid")
8 sns.set_style("ticks")
9
10 Sigma = np.array([
11     [2,1,1,1],
12     [1,2,1,1],
13     [1,1,2,1],
14     [1,1,1,2]
15 ])
16
17 sqrtSigma = np.array([
18     [1.309, 0.309, 0.309, 0.309],
19     [0.309, 1.309, 0.309, 0.309],
20     [0.309, 0.309, 1.309, 0.309],
21     [0.309, 0.309, 0.309, 1.309]
22 ])
23
24
25 #Arquitectura de la red
26 input_size = 4
27 output_size = 1
28 learning_rate = 0.01
29 num_epochs = 250
30 fig_index = "A"
31 epochs = [i for i in range(0, num_epochs)]
32
33 w_plot = np.zeros((num_epochs, input_size))
34 dot_plot = np.zeros((num_epochs, input_size))
35
36 # Pesos iniciales aleatorios
37 w = np.random.uniform(-0.1,0.1,size=(output_size, input_size)) #filas #columnas
38
39 autovalores, autovectores = np.linalg.eig(Sigma)
40
41 print("Autovalores de Sigma:")
42 print(autovalores)
43 print("Autovectores de Sigma:")
44 print((autovectores.T[1]).reshape(-1, 1))
45 print("\n")
46
47 for epoch in tqdm(range(num_epochs)):
48
49     w_plot[epoch] = w
50     z_train = np.random.multivariate_normal([0,0,0,0], Sigma)
51     x_train = np.dot(sqrtSigma,z_train)
52
53     O = np.dot(w, x_train)
54
55     for i in range(input_size):
56         dot_plot[epoch][i] = np.dot(w, autovectores.T[i].reshape(-1, 1))
57
58     # Actualizacion de pesos
59     w += learning_rate*O*(x_train-O*w)
60
61 print("Entrenamiento completado.")
62
63 fig1, ax1 = plt.subplots(figsize=(8,6))
64 for i in range(input_size):
65     ax1.plot(epochs, w_plot[:,i], "-", label = r"$w_{" + str(i) + "}$")
66     ax1.hlines(y=0.5, xmin=0, xmax=num_epochs, linestyle='--', color='gray')
67     ax1.hlines(y=-0.5, xmin=0, xmax=num_epochs, linestyle='--', color='gray')
68     ax1.set_xlabel(r"Epoca", fontsize=18)
```

```

69 ax1.set_ylabel(r"Pesos $w_j$", fontsize=18)
70 ax1.tick_params(direction='in', top=True, right=True, left=True, bottom=True)
71 ax1.tick_params(axis='x', rotation=0, labelsize=18, color='black')
72 ax1.tick_params(axis='y', labelsize=18, color='black')
73 ax1.legend(fontsize=12, framealpha=1, loc= "center right")
74 ax1.set_xlim(0, num_epochs)
75 ax1.text(0.05, 0.9, f'{fig_index}', transform=ax1.transAxes, fontsize=24, verticalalignment='top',
76         fontweight='bold', color="black")
77 ax1.grid(True, linewidth=0.5, linestyle='--', alpha=0.9)
78 fig1.savefig(f"../Redes-Neuronales/Practica_5/resultados/ej1/pesos_lr_{learning_rate}.pdf")
79 fig1.savefig(f"../Redes-Neuronales/Practica_5/resultados/ej1/pesos_lr_{learning_rate}.png", dpi=600)
80
81 fig2, ax2 = plt.subplots(figsize=(8,6))
82 for i in range(input_size):
83     ax2.plot(epochs, dot_plot[:, i], "--", label=r"$\vec{w} \cdot \vec{v}_i$")
84 ax2.hlines(y=1.0, xmin=0, xmax=num_epochs, linestyle='--', color='gray')
85 ax2.hlines(y=-1.0, xmin=0, xmax=num_epochs, linestyle='--', color='gray')
86 ax2.hlines(y=0.0, xmin=0, xmax=num_epochs, linestyle='--', color='gray')
87 ax2.set_xlabel(r"Epoca", fontsize=18)
88 ax2.set_ylabel(r"Producto escalar $\vec{w} \cdot \vec{v}_j$", fontsize=18)
89 ax2.tick_params(direction='in', top=True, right=True, left=True, bottom=True)
90 ax2.tick_params(axis='x', rotation=0, labelsize=18, color='black')
91 ax2.tick_params(axis='y', labelsize=18, color='black')
92 ax2.legend(fontsize=12, framealpha=1, loc= "upper right")
93 ax2.set_xlim(0, num_epochs)
94 ax2.text(0.05, 0.9, f'{fig_index}', transform=ax2.transAxes, fontsize=24, verticalalignment='top',
95         fontweight='bold', color="black")
96 ax2.grid(True, linewidth=0.5, linestyle='--', alpha=0.9)
97 fig2.savefig(f"../Redes-Neuronales/Practica_5/resultados/ej1/prodesc_lr_{learning_rate}.pdf")
98 fig2.savefig(f"../Redes-Neuronales/Practica_5/resultados/ej1/prodesc_lr_{learning_rate}.png", dpi=600)

```

B. Ejercicio 2

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from tqdm import tqdm
4 #Ploteo
5 import seaborn as sns
6 import matplotlib.patches as patches
7 #sns.axes_style("whitegrid")
8 sns.set_style("ticks")
9
10 def gaussian(i, i_, sigma):
11     return np.exp(-(((i-i_)**2)/(2*(sigma**2))))
12
13 def input_semicircle():
14     f = True
15     while f:
16         x = np.random.uniform(-1.1,1.1)
17         y = np.random.uniform(0,1.1)
18
19         r = np.linalg.norm([x,y])
20         if( 0.9 <= r <= 1.1 and 0 <= np.arctan2(y,x) <= np.pi ):
21             f = False
22     return np.array([x,y])
23
24 def input_triangle():
25     f = True
26     while f:
27         x = np.random.uniform(-0.5,0.5)
28         y = np.random.uniform(0.0,np.sqrt(3)/2)
29
30         if ( -0.5 <= x <= 0.5 and 0 <= y <= np.tan(np.pi/3)*(0.5-np.abs(x)) ):
31             f = False
32     #ax1.plot(x, y, marker='o', markersize=1, color='k')
33     return np.array([x,y])
34
35 def input_square():
36     f = True
37     while f:
38         x = np.random.uniform(-0.5,0.5)

```

```
39     y = np.random.uniform(0.0,2.0)
40     # Verifica si el punto esta en la mitad inferior del cuadrado
41     if (-0.5 <= x <= 0.5) and (0.0 <= y <= 1.0):
42         # Genera un numero aleatorio adicional para duplicar la probabilidad en la mitad inferior
43         if y <= 0.5:
44             if np.random.rand() <= 0.5:
45                 f = False
46             else:
47                 f = True
48     #ax1.plot(x, y, marker='o', markersize=0.5, color='k')
49     return np.array([x,y])
50
51 #Arquitectura de la red
52 input_size = 2
53 output_size = 10
54 learning_rate = 0.01
55 sigma_0 = 10000
56 #sigma = 1.0
57 num_epochs = 100000
58 fig_index = "C"
59 epochs = [i for i in range(0, num_epochs)]
60
61 w_plot = np.zeros((num_epochs, input_size))
62
63 w = np.array([[np.random.uniform(-0.75, 0.75), np.random.uniform(-0.25, 1.25)] for _ in range(output_size)])
64 """
65 w = np.array([[ 0.1389785 ,  0.02162108],
66 [ 0.00636808 ,  0.16040874],
67 [ 0.05046116 ,  0.0006476 ],
68 [ 0.15113858 ,  0.04233544],
69 [ 0.08499907 ,  0.13616288],
70 [-0.09924555 ,  0.15332305],
71 [-0.00520648 ,  0.08107896],
72 [-0.18375063 ,  0.15545262],
73 [-0.03089037 ,  0.15672824],
74 [-0.06498289 ,  0.07104595]])
75 """
76
77 plot_w = np.zeros((num_epochs, output_size, input_size))
78
79 fig1, ax1 = plt.subplots()
80
81 for i in range (output_size):
82     ax1.plot(w[i, 0], w[i, 1], 'o', color='k', markersize = 3)
83
84 for epoch in tqdm(range(num_epochs)):
85     x = input_square()
86     norm = np.linalg.norm(w - x, axis=1)
87     i_ = np.argmin(norm)
88     sigma = sigma_0/(epoch+1)
89     # Actualizacion de pesos
90     for i in range(output_size):
91         w[i] += learning_rate*gaussian(i,i_,sigma)*(x-w[i])
92         plot_w[epoch][i] = w[i]
93
94 for i in range ( output_size ):
95     ax1.text(w[i, 0], w[i, 1], str(i), fontsize=15, ha='center', va='center', color='k')
96
97
98 for i in range(output_size):
99     x_trajectory = plot_w[:, i, 0]
100    y_trajectory = plot_w[:, i, 1]
101    plt.plot(x_trajectory, y_trajectory, label=f"Neurona {i+1}")
102
103 print("Entrenamiento completado.")
104
105 ax1.set_xlabel(r"$x$)", fontsize=18)
106 ax1.set_ylabel(r"$y$)", fontsize=18)
107 ax1.tick_params(direction='in', top=True, right=True, left=True, bottom=True)
108 ax1.tick_params(axis='x', rotation=0, labelsize=18, color='black')
```

```
110 ax1.tick_params(axis='y', labelsize=18, color='black')
111 #ax1.legend(fontsize=12, framealpha=1)
112 #ax1.set_xlim(-1.2, 1.2)
113 #ax1.set_ylim(-0.1, 1.2)
114 ax1.text(0.05, 0.95, f'{fig_index}', transform=ax1.transAxes, fontsize=24, verticalalignment='top',
115     fontweight='bold', color="black")
116 ax1.grid(True, linewidth=0.5, linestyle='-', alpha=0.9)
117 ax1.set_aspect('equal')
118
119 #ax1.plot(0, 0.6387, marker='x', markersize=10, color='k', label='Centro de masa')
120 ring = patches.Wedge(center=(0, 0), r=1.1, theta1=0, theta2=180, width=0.2, facecolor='lightgray',
121     edgecolor='black', alpha=0.5)
122 ax1.add_patch(ring)
123 ax1.set_xlim(-1.5, 1.5)
124 ax1.set_ylim(-0.5, 1.5)
125
126 #triangle = patches.Polygon(np.array([[-0.5, 0], [0.5, 0], [0, np.sqrt(3) / 2]]), closed=True, facecolor='
127     lightgray', edgecolor='black')
128 #ax1.add_patch(triangle)
129 #ax1.plot(0, 0.28867, marker='x', markersize=15, color='k', label='Centro de masa')
130 #ax1.set_xlim(-0.75, 0.75)
131 #ax1.set_ylim(-0.75, 1.25)
132
133 #square = patches.Rectangle((-0.5, 0), 1, 1, angle=0, facecolor='lightgray', edgecolor='black')
134 #ax1.add_patch(square)
135 #ax1.plot(0, 0.583, marker='x', markersize=15, color='k', label='Centro de masa')
136 #ax1.set_xlim(-0.75, 0.75)
137 #ax1.set_ylim(-0.25, 1.25)
138
139 fig1.savefig(f'../Redes-Neuronales/Practica_5/resultados/ej2/pesos_sigma_{sigma}_epochs_{num_epochs}_lr_{
140     learning_rate}.pdf')
141 fig1.savefig(f'../Redes-Neuronales/Practica_5/resultados/ej2/pesos_sigma_{sigma}_epochs_{num_epochs}_lr_{
142     learning_rate}.png", dpi=600)
```