

Aprendizaje no supervisado

Pablo Chehade

pablo.chehade@ib.edu.ar

Redes Neuronales, Instituto Balseiro, CNEA-UNCuyo, Bariloche, Argentina, 2023

EJERCICIO 1

Se entrenó de manera no supervisada una red neuronal lineal de una sola capa con cuatro entradas y una salida. Los datos de entrada presentan una distribución gaussiana con matriz de correlación Σ

$$\Sigma = \begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{bmatrix}.$$

El autovector asociado al mayor autovalor de Σ es $\vec{v} = \frac{1}{2}(1, 1, 1, 1)$.

Se partió de un vector de pesos $\vec{w} = (w_1, w_2, w_3, w_4)$, con cada componente dada por una distribución uniforme con máximo 0,01. Luego, se aplicó la regla de Oja

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

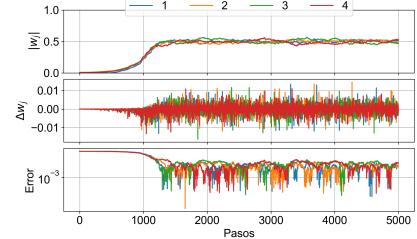
Aquí, η representa la tasa de aprendizaje, V es la salida de la red y ξ_j es la componente j del dato de entrada $\vec{\xi}$. En base a la teoría desarrollada para la regla de Oja, se espera que el vector de pesos \vec{w} tienda al autovector \vec{v} .

En la figura 1a, se ilustra la evolución del módulo de los pesos $|w_j|$, las modificaciones Δw_j y la diferencia entre w_j y v_j , para $\eta = 0,001$. Tras un período transitorio, se observa que las componentes de \vec{w} tienden hacia el valor 0,5, indicando convergencia hacia \vec{v} . Dependiendo de las condiciones iniciales, el vector \vec{w} tiende a alinearse en la dirección $+\vec{v}$ o $-\vec{v}$. Debido a esto se graficó el módulo de cada componente. Posteriormente, en estado estacionario, los pesos continúan modificándose. Por ello, en la figura 1b, se procesan los datos anteriores realizando un promedio en cada paso de tiempo de los 200 pasos adyacentes. Esto resulta en un comportamiento más suave y una reducción notable en las variaciones de Δw_j de hasta un orden de magnitud. Luego de este procesamiento de los datos se confirma nuevamente que \vec{w} tiende a \vec{v} .

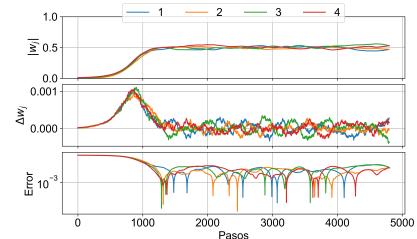
Por último, en la figura 1c, se evolucionó el sistema con $\eta = 0,01$. El sistema muestra una convergencia más rápida y variaciones Δw_j de mayor magnitud. Además, se determinó que existe un valor superior límite para η , más allá del cual el sistema tiende a diverger.

EJERCICIO 2

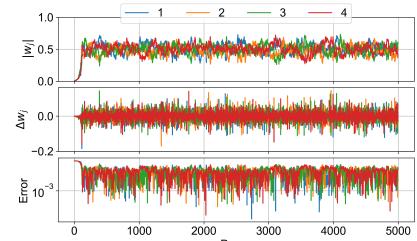
Se entrenó una red neuronal de Kohonen con dos neuronas de entrada y diez neuronas de salida alineadas en



(a)



(b)



(c)

Figura 1: Evolución del módulo de los pesos w_j , las modificaciones Δw_j y la diferencia entre w_j y v_j durante el entrenamiento de la red neuronal. En a se empleó una tasa de aprendizaje de $\eta = 0,001$. En b se empleó la misma tasa de aprendizaje pero además se procesaron los datos promediando en cada paso de tiempo sobre los 200 pasos adyacentes. En c se empleó una tasa de aprendizaje de $\eta = 0,01$.

una línea. La red se alimenta mediante una distribución de entrada definida como:

$$P(\xi) = P(r, \theta) = \begin{cases} \text{constante} & \text{si } r \in [0, 9, 1, 1], \theta \in [0, \pi] \\ 0 & \text{si no,} \end{cases}$$

donde r y θ son las coordenadas polares del vector $\vec{\xi}$.

Los pesos iniciales w_{ij} , donde i representa la neurona de salida y j la dimensión del espacio de entrada, se asignan aleatoriamente con una distribución uniforme con un

valor máximo de 0.1. En cada iteración del entrenamiento, se selecciona un vector de entrada $\vec{\xi}$, y se identifica la neurona "ganadora" i^* como aquella que minimiza la distancia euclídea con el vector de entrada, es decir

$$i^* = \operatorname{argmin}_i |\vec{w}_i - \vec{\xi}|,$$

donde $\vec{w}_i = (w_{i1}, w_{i2})$. Posteriormente, se actualizan los pesos de todas las neuronas utilizando la regla

$$\Delta \vec{w}_i = \eta \Gamma(i, i^*) (\vec{\xi} - \vec{w}_i),$$

donde

$$\Gamma(i, i^*) = \exp\left(-\frac{(i - i^*)^2}{2\sigma^2}\right)$$

es la función de vecindad y η es la tasa de aprendizaje.

Se realizó un análisis sobre la evolución de los pesos de la red variando los parámetros η y σ , y se observó cómo afectan la convergencia, haciendo incapié cómo se modifican los pesos durante el entrenamiento.

Se observaron diferentes comportamientos al variar los parámetros η y σ . En todos los casos se estudió la convergencia de los pesos \vec{w}_i mediante su evolución graficada en el espacio de los datos de entrada (figura 2)

1. $\eta = 0,1$ y $\sigma = 1$: Inicialmente los pesos \vec{w}_i son cercanos entre sí. Luego se desplazan hacia las regiones donde $P(\vec{\xi})$ es no nula. Finalmente, los pesos se organizan sobre la región, organizados de acuerdo a la disposición de las neuronas sobre la línea.

2. $\eta = 0,1$ y $\sigma = 2$: En este caso, los pesos se mueven más juntos en comparación con el caso anterior debido a un mayor valor de σ , que incrementa el efecto de la función de vecindad. Esto se puede entender más claramente en el caso extremo en el que $\sigma \rightarrow \infty$. En tal caso, $\Gamma \rightarrow 1$ y todas las neuronas se actualizan en la misma magnitud hacia la posición del dato de entrada $\vec{\xi}$. En cuanto al estado final, incluso con una gran cantidad de pasos de entrenamiento no se llega al mismo estado que antes. Una forma de resolver este problema podría ser aprovechando el comportamiento visto en el caso anterior y disminuir progresivamente σ a medida que se realiza el entrenamiento.

3. $\eta = 0,01$ y $\sigma = 1$: Con una tasa de aprendizaje reducida, la evolución es más lenta. Además, los pesos muestran variaciones menores durante el entrenamiento. A pesar de la lentitud, con suficientes iteraciones el sistema converge a la misma condición que el primer caso.

El estudio sugiere que la elección de η y σ es crucial. Un η más bajo lleva a una convergencia más lenta pero con menor ruido, mientras que un σ más alto promueve una adaptación global pero menos detallada de los pesos.

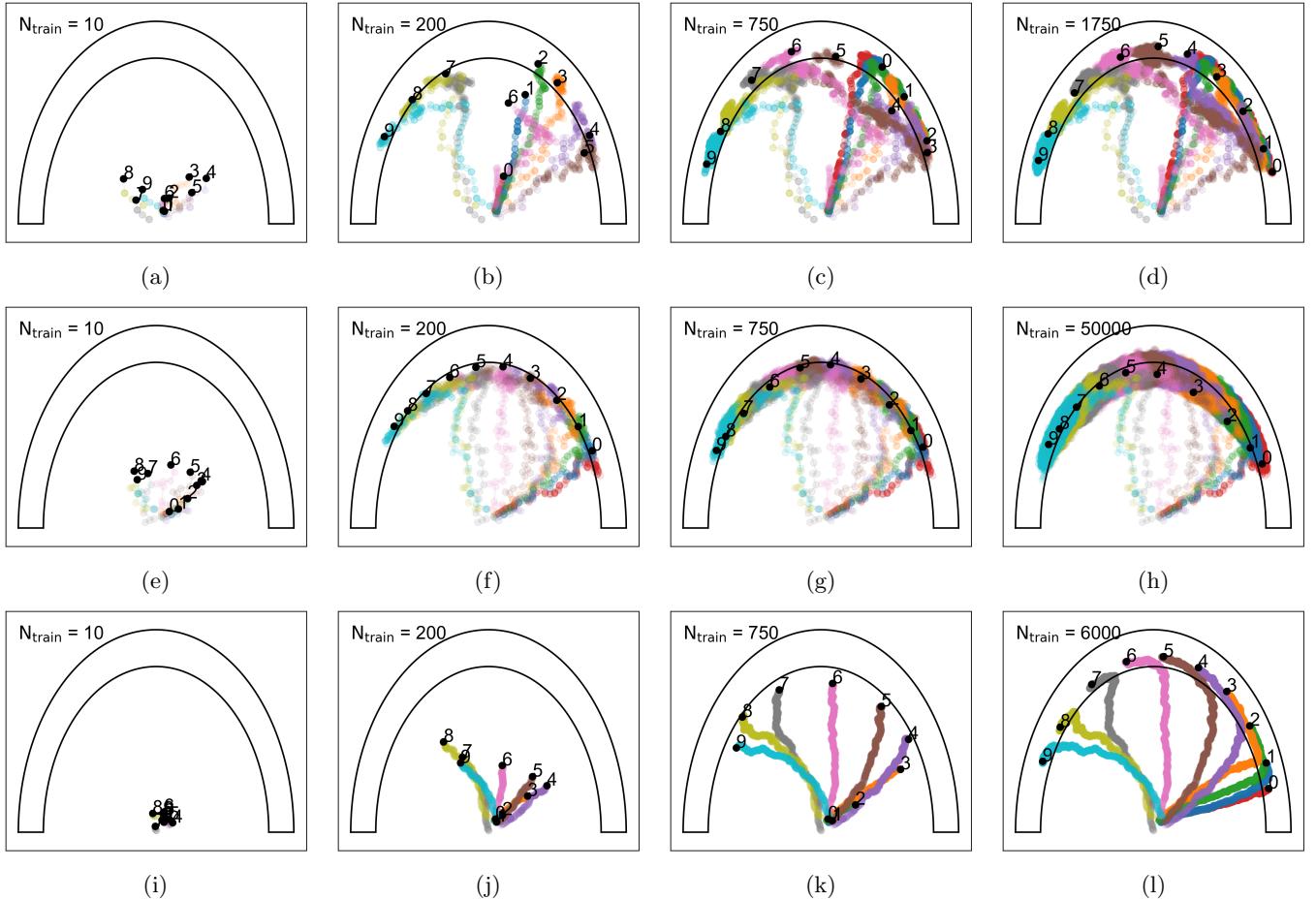


Figura 2: Evolución de los peso \vec{w}_i con $i = 1, 2, 3, \dots, 10$ en el espacio de los datos de entrada para distintos pasos de entrenamiento N_{train} . Se empleó a-d $\eta = 0,1$, $\sigma = 1$, e-h $\eta = 0,1$, $\sigma = 2$, i-l $\eta = 0,01$, $\sigma = 1$. La región semicircular corresponde a la región donde se encuentran los datos de entrada. En negro se grafica la posición de los pesos finales para cada valor de N_{train} , junto al número correspondiente de neurona. La numeración de las neuronas se realiza en base a que se encuentran dispuestas en una línea. En color se grafican las posiciones previas. No se grafican los ejes por simplicidad.

APÉNDICE

A continuación se desarrolla el código empleado durante este trabajo implementado en Python.

```

1  #### Ejercicio 1
2  #Import libraries
3  import numpy as np
4  import matplotlib
5  import matplotlib.pyplot as plt
6
7
8  #Parameters
9  w_ini_max = 0.01 #valor maximo del peso inicial
10
11 #Def correlation matrix
12 C = np.array([[2, 1, 1, 1],
13               [1, 2, 1, 1],
14               [1, 1, 2, 1],
15               [1, 1, 1, 2]])
16

```

```

17 C_12 = np.array([[1.309, 0.309, 0.309, 0.309],
18                 [0.309, 1.309, 0.309, 0.309],
19                 [0.309, 0.309, 1.309, 0.309],
20                 [0.309, 0.309, 0.309, 1.309]])
21
22 #Calculo los autovalores y autovectores de C
23 eigvals, eigvecs = np.linalg.eig(C)
24 #Calculo el autovector con el mayor autovalor
25 eigvec_max = eigvecs[:, np.argmax(eigvals)]
26 #Imprimo
27 print('El autovector con el mayor autovalor es:', eigvec_max)
28
29 def ejemplo():
30     #Genero 4 nros con distribucion normal
31     z = np.random.randn(4)
32     #Multiplico por C_12
33     x = np.dot(C_12, z)
34     return x
35
36 def inicializacion_pesos():
37     #Genero 4 nros con distribucion uniforme
38     w = np.random.uniform(-w_ini_max, w_ini_max, 4)
39     return w
40
41
42 def actualizacion_pesos(w, eta):
43     #Genero un ejemplo
44     xi = ejemplo()
45     #Calculo la salida
46     V = np.dot(w, xi)
47     #Calculo delta_w
48     delta_w = eta*V*(xi - V*w)
49     return w + delta_w
50
51 def train(N_train, eta):
52     #Genero la matriz de pesos
53     w_matrix = np.zeros([N_train, 4])
54     #Genero los pesos iniciales
55     w = inicializacion_pesos()
56     w_matrix[0] = w
57     #Entreno N_train pasos
58     for i in range(1,N_train):
59         w = actualizacion_pesos(w, eta)
60         w_matrix[i] = w
61     return w_matrix
62
63 #Entreno una red
64 N_train = 5000
65
66 #Promedio w_matrix y delta_w para cada iindice en una caja de ancho M con np.convolve
67 def plt_w(M, eta):
68     w_matrix = train(N_train, eta)
69     #Calculo los delta_w
70     delta_w = w_matrix[1:] - w_matrix[:-1]
71
72     w_matrix_mean = np.empty([N_train - M + 1, 4])
73     delta_w_mean = np.empty([N_train - M, 4])
74     for i in range(4):
75         w_matrix_mean[:, i] = np.convolve(w_matrix[:, i], np.ones(M)/M, mode='valid')
76         delta_w_mean[:, i] = np.convolve(delta_w[:, i], np.ones(M)/M, mode='valid')
77
78     # Grafico los pasos en el tiempo y en otra grafica, los cambios delta w_j entre
79     # pasos
80     fig, ax = plt.subplots(3, 1, figsize=(9.5, 5.5), sharex=True)

```

```

80     fig.subplots_adjust(hspace=0.1)
81
82     ax[0].plot(np.abs(w_matrix_mean))
83     # ax[0].set_xlabel('Pasos')
84     ax[0].set_ylabel('$|w_j|$')
85     ax[0].labels = ['1', '2', '3', '4']
86     #Grafico las labels arriba fuera del grafico
87     ax[0].legend(ax[0].labels, loc='upper center', bbox_to_anchor=(0.5, 1.3), ncol=4)
88     ax[0].set_ylim(0, 1)
89     # ax[0].set_xscale("log")
90     ax[0].grid()
91
92     ax[1].plot(delta_w_mean)
93     # ax[1].set_xlabel('Pasos')
94     ax[1].set_ylabel('$\Delta w_j$')
95     ax[1].labels = ['1', '2', '3', '4']
96     # ax[1].legend(ax[1].labels)
97     ax[1].grid()
98
99     ax[2].plot(np.abs(np.abs(w_matrix_mean) - np.abs(eigvec_max)))
100    ax[2].set_xlabel('Pasos')
101    ax[2].set_ylabel('Error')
102    ax[2].labels = ['1', '2', '3', '4']
103    #Grafico las labels arriba fuera del grafico
104    # ax[2].legend(ax[2].labels)
105    ax[2].set_yscale("log")
106    ax[2].grid()
107
108    plt.show()
109
110 #Hago graficos para el informe
111 plt_w(1, 0.001)
112 plt.savefig('Informe/ej1_fig1.png', bbox_inches='tight', dpi=300)
113
114 plt_w(200, 0.001)
115 plt.savefig('Informe/ej1_fig2.png', bbox_inches='tight', dpi=300)
116
117 plt_w(1, 0.01)
118 plt.savefig('Informe/ej1_fig3.png', bbox_inches='tight', dpi=300)
119
120 # ## Ejercicio 2
121 #Def parametros globales
122 r1 = 0.9
123 r2 = 1.1
124 N = 2
125 M = 10
126 w_ini_max = 0.1
127
128 #Def seed
129 np.random.seed(0)
130
131 #Genero pares de nros aleatorios (x,y) con distribucion uniforme sobre un circulo
132 def generar_datos(n):
133     r_array = np.empty([n,2])
134     contador = 0
135     while(contador < n):
136         #Genero nros aleatorios con distribucion uniforme entre 0 y 1
137         x = (np.random.rand()*2 - 1)*r2
138         y = (np.random.rand()*2 - 1)*r2
139         #Me fijo si pertenece al anillo
140         r = np.sqrt(x**2 + y**2)
141         theta = np.arctan2(y,x)
142         if r1 < r < r2 and 0 < theta < np.pi:
143             r_array[contador] = np.array([x,y])

```

```

144         contador += 1
145     return r_array
146
147     def Vecindad(i,i_star,sigma):
148         return np.exp(-((i-i_star)**2)/(2*sigma**2))
149
150     def algoritmo_de_Kohonen(w, x, sigma, eta):
151         #Calculo la neurona ganadora
152         i_star = np.argmin(np.linalg.norm(w-x, axis=1))
153         #Actualizo los pesos
154         delta_w = eta*Vecindad(np.arange(M),i_star,sigma).reshape(M,1)*(x-w)
155         w += delta_w
156     return w
157
158     def entrenamiento_Kohonen(N_train, sigma, eta):
159         #Inicializo pesos de la red
160         w = np.random.rand(M,N)*w_ini_max
161         w_matriz = np.empty([N_train,M,N])
162         #Genero los datos
163         r_array = generar_datos(N_train)
164         #Entreno la red
165         for i in range(N_train):
166             w = algoritmo_de_Kohonen(w, r_array[i], sigma, eta)
167             #Guardo el valor
168             w_matriz[i] = w
169     return w_matriz
170
171     def plt_entrenamiento_Kohonen(N_train, sigma, eta):
172         #Redef seed
173         np.random.seed(0)
174         w_matriz = entrenamiento_Kohonen(N_train, sigma, eta)
175
176         #Graph los pesos en el tiempo
177         fig, ax = plt.subplots(figsize = (5,4))
178         color_vec = ["tab:red", "tab:blue", "tab:green", "tab:orange", "tab:purple", "tab:brown",
179                         "tab:pink", "tab:gray", "tab:olive", "tab:cyan"]
180
181         #Grafico los pesos
182         for i in range(M):
183             ax.plot(w_matriz[0:,i,0],w_matriz[0:,i,1], 'o-', color=color_vec[i], alpha =
184                 0.1)
185         for i in range(M):
186             #Pinto de otro color el ultimo punto
187             ax.plot(w_matriz[-1,i,0],w_matriz[-1,i,1], 'o-', color="k", alpha = 1)
188             ax.text(w_matriz[-1,i,0],w_matriz[-1,i,1], str(i), color="k", alpha=1)
189
190         #Grafico un anillo entre r1 y r2 con theta entre 0 y pi
191         theta = np.linspace(0,np.pi,100)
192         ax.plot(r1*np.cos(theta),r1*np.sin(theta), 'k')
193         ax.plot(r2*np.cos(theta),r2*np.sin(theta), 'k')
194         ax.plot([-r2,-r1],[0,0], 'k'); ax.plot([r1,r2],[0,0], 'k')
195         ax.set_xlim(-1.2,1.2)
196         ax.set_ylim(-0.1,1.2)
197
198         #Quito los tickslabels
199         ax.set_xticklabels([])
200         ax.set_yticklabels([])
201         ax.set_xticks([])
202         ax.set_yticks([])
203
204         #Agrego arriba a la izquierda el valor de N_train
205         ax.text(-1.1,1.05, '$\mathbf{N_{train}}$=' + str(N_train), color="k", alpha=1)
206
207         plt.show()
208     return

```

```
206 #Guardo graficos particulares y los guardo
207 # plt_entrenamiento_Kohonen(N_train, sigma, eta)
208
209 #Figura 1
210 sigma = 1
211 eta = 0.1
212
213
214 plt_entrenamiento_Kohonen(10, sigma, eta)
215 plt.savefig('Informe/ej2_fig1_1.png', bbox_inches='tight', dpi=300)
216
217 #Descomentar las siguientes lineas para hacer la variacion de N_train, eta y sigma
218 # plt_entrenamiento_Kohonen(200, sigma, eta)
219 # plt.savefig('Informe/ej2_fig1_2.png', bbox_inches='tight', dpi=300)
220
221 # plt_entrenamiento_Kohonen(750, sigma, eta)
222 # plt.savefig('Informe/ej2_fig1_3.png', bbox_inches='tight', dpi=300)
223
224 # plt_entrenamiento_Kohonen(1750, sigma, eta)
225 # plt.savefig('Informe/ej2_fig1_4.png', bbox_inches='tight', dpi=300)
226
227 # #Figura 2
228 # sigma = 2
229 # eta = 0.1
230
231 # plt_entrenamiento_Kohonen(10, sigma, eta)
232 # plt.savefig('Informe/ej2_fig2_1.png', bbox_inches='tight', dpi=300)
233
234 # plt_entrenamiento_Kohonen(200, sigma, eta)
235 # plt.savefig('Informe/ej2_fig2_2.png', bbox_inches='tight', dpi=300)
236
237 # plt_entrenamiento_Kohonen(750, sigma, eta)
238 # plt.savefig('Informe/ej2_fig2_3.png', bbox_inches='tight', dpi=300)
239
240 # plt_entrenamiento_Kohonen(50000, sigma, eta)
241 # plt.savefig('Informe/ej2_fig2_4.png', bbox_inches='tight', dpi=300)
242
243 # #Figura 3
244 # sigma = 1
245 # eta = 0.01
246
247 # plt_entrenamiento_Kohonen(10, sigma, eta)
248 # plt.savefig('Informe/ej2_fig3_1.png', bbox_inches='tight', dpi=300)
249
250 # plt_entrenamiento_Kohonen(200, sigma, eta)
251 # plt.savefig('Informe/ej2_fig3_2.png', bbox_inches='tight', dpi=300)
252
253 # plt_entrenamiento_Kohonen(750, sigma, eta)
254 # plt.savefig('Informe/ej2_fig3_3.png', bbox_inches='tight', dpi=300)
255
256 # plt_entrenamiento_Kohonen(6000, sigma, eta)
257 # plt.savefig('Informe/ej2_fig3_4.png', bbox_inches='tight', dpi=300)
```