# Práctica 5 - Aprendizaje no supervisado

Zablotsky, Amir Nicolás
*Redes Neuronales 2022*
*Instituto Balseiro, UNCuyo*

## EJERCICIO 1

En este ejercicio se estudió el proceso de aprendizaje no supervisado en una red con una capa lineal de cuatro entradas y una única salida. La ecuación de salida viene dada por la Ecuación 1, donde $\xi_j$ representa la entrada de cada neurona y $w_j$ el peso asociado a su conexión con la salida.

$$V = \sum_{j=1}^{4} \xi_j w_j = \vec{\xi} \cdot \vec{w} \tag{1}$$

La distribución de las entradas de la red corresponde a una Gaussiana multivariable con matriz de correlación $\Sigma$ (Ec. 2).

$$P(\vec{\xi}) = \frac{1}{(2\pi)^2 \sqrt{\det(\Sigma)}} \exp\left(-\frac{1}{2}\vec{\xi}^T \Sigma^{-1} \vec{\xi}\right) , \quad \Sigma = \begin{pmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{pmatrix} \tag{2}$$

Para llevar a cabo el proceso de aprendizaje no supervisado, se partió de un vector de pesos iniciales aleatorio $\vec{w}_0$, y se utilizó la regla de Oja (Ec. 3) para actualizar los pesos a lo largo de las épocas, con una tasa de aprendizaje $\eta = 10^{-4}$.

$$\Delta w_j = \eta V(\xi_j - V w_j) \tag{3}$$

Al emplear la regla de Oja, el problema tiene una única solución estable $\vec{w}$ paralela al autovector de la matriz $\Sigma$ asociado al máximo autovalor. Los autovalores de la matriz $\Sigma$ son $\lambda_1 = \lambda_3 = \lambda_4 = 1$, y $\lambda_2 = 5$. El autovector asociado al autovalor $\lambda_{\max} = 5$ es $\vec{v}_{\max} = \frac{1}{2}(1, 1, 1, 1)^T$.

Como podemos ver en la Figura 1, las cuatro componentes del vector $\vec{w}$ tienden asintóticamente a $\frac{1}{2}$, es decir se alinean con $\vec{v}_{\max}$. Más aun, podemos ver como $\|\vec{w}\| \to 1$, lo cual es otra característica de la regla de Oja.

En la Figura 2 se ilustra la similitud entre el vector $\vec{w}$ y cada uno de los autovectores, dada por

$$\mathrm{Sim}(\vec{w}, \vec{v}_j) = \frac{|\vec{w} \cdot \vec{v}_j|}{\|\vec{w}\| \|\vec{v}_j\|} ,$$

en función de las épocas. Como se puede ver, la similitud entre el vector de pesos y el autovector asociado al mayor autovalor tiende a 1, mientras que su similitud con los demás vectores tiende a 0. Esto se debe a que la matriz $\Sigma$ es hermitiana, por lo que todos sus autovectores son ortogonales entre si, y por ende ortogonales a $\vec{w}$ (a excepción del autovector asociado al máximo autovalor).

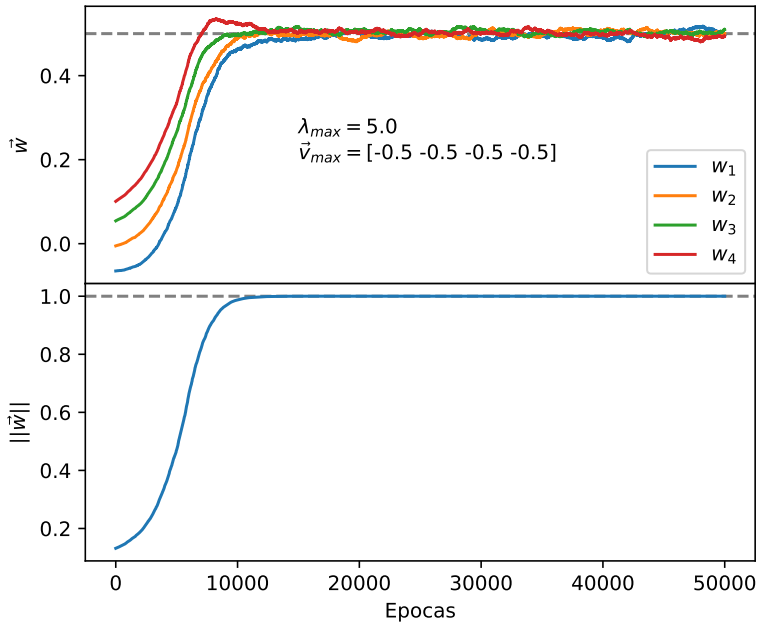


Figura 1: (Arriba) Valor de las componentes $w_j$ en función de las épocas, según la regla de Oja. (Abajo) Norma del vector de pesos $\vec{w}$ en función de las épocas.

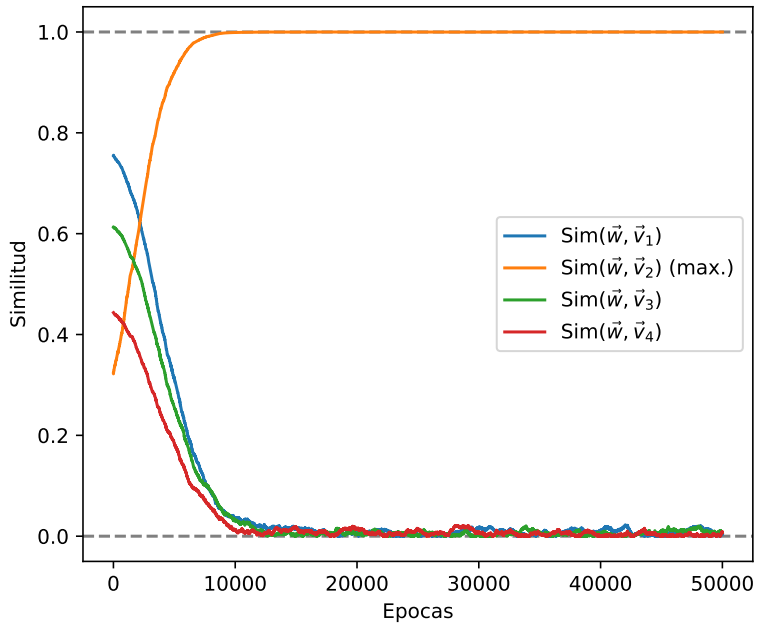


Figura 2: Similitud del vector de pesos con cada uno de los autovectores de la matriz de correlación $\Sigma$.

## EJERCICIO 2

En este ejercicio se estudió el proceso de aprendizaje no supervisado competitivo en una red de Kohonen con dos neuronas de entrada y diez neuronas de salida, dispuestas de manera lineal. Inicialmente, las neuronas de salida son dispuestas de forma equiespaciada entre los puntos $(-1, 1/2)$ y $(1, 1/2)$.

La distribución de probabilidad de las entradas de la red está dada por la Ecuación 4, es decir probabilidad uniforme sobre una corona semicircular de ancho 0,2. En la Figura. 3 se observa la distribución de los datos de entrada generados.

$$P(r,\theta) = \begin{cases} \text{cte, si } r \in [0{,}9, 1{,}1]\,,\ \theta \in [0,\pi] \\ 0\text{, si no.} \end{cases} \tag{4}$$

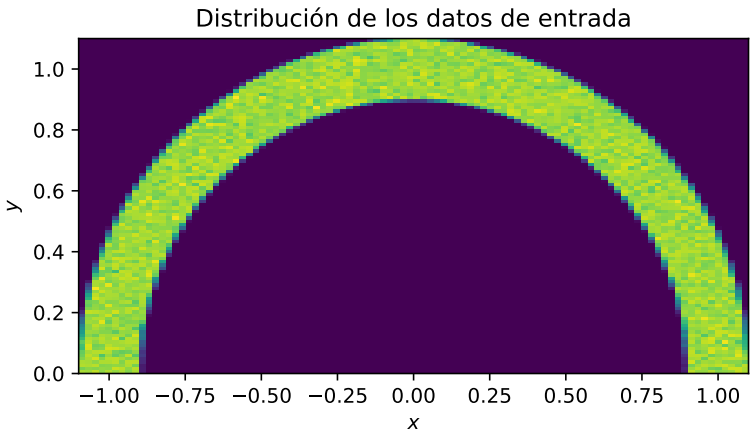


Figura 3: Distribución de los datos de entrada de la red, generados según la Ec. 4.

La actualización de los pesos ($\vec{w} \in \mathbb{R}^{2\times 10}$) en cada época se realiza mediante la regla

$$\Delta\vec{w}_i = \eta\Lambda(i,i^*)(\vec{\xi} - \vec{w}_i),\ \text{con } \Lambda(i,i^*) \propto \exp\left(-\frac{(i-i^*)^2}{2\sigma^2}\right),$$

donde $\vec{w}_i$ corresponde los vectores de pesos de cada una de las 10 neuronas de salida. La función de vecindad $\Lambda(i,i^*)$ introduce un efecto de localidad espacial, que se vuelve más local cuanto más chico es el $\sigma$.

En la Figura 4 se puede ver la evolución de los pesos luego de 3125, 12500 y 50000 épocas, con una tasa de aprendizaje $\eta = 10^{-3}$, utilizando funciones de vecindad con $\sigma \in \{0.1, 0.5, 2, 5\}$. A modo de referencia, el área sombreada indica la región de la cual se extraen los datos de entrada según la Ecuación 4.

En el caso de $\sigma = 0.1$ vemos que, incluso para una gran cantidad de épocas, los pesos no alcanzan a converger de manera correcta sobre la región de los datos de entrada. Esto se debe a que la función de vecindad es altamente local, por lo que en cada iteración tiende a actualizarse solamente el peso más cercano a los datos, que debido a la alta localidad, suelen ser los que ya han sido acercados a la zona sombreada previamente. Debido a esto, algunos pesos logran converger a la zona sombrada, mientras que otros quedan cercanos a su posición inicial. Por otro lado, cuando $\sigma = 2$ o $\sigma = 5$ la función de vecindad es muy ancha, por lo cual en cada iteración se actualizan varios (o todos) los pesos de la misma forma. Esto causa que a la red se le dificulte captar la estructura fina de los datos de entrada, y puede converger a estructuras similares a los datos de entrada pero con menor detalle (por ejemplo un arco más pequeño). Finalmente, se observa que con $\sigma = 0.5$ los pesos convergen correctamente a la zona de los datos de entrada, ya que la función de vecindad no es ni tan local como para "olvidarse" algunos de los pesos, ni tan ancha como para actualizar muchos pesos en cada iteración y de esta forma perder detalle de los datos de entrada.

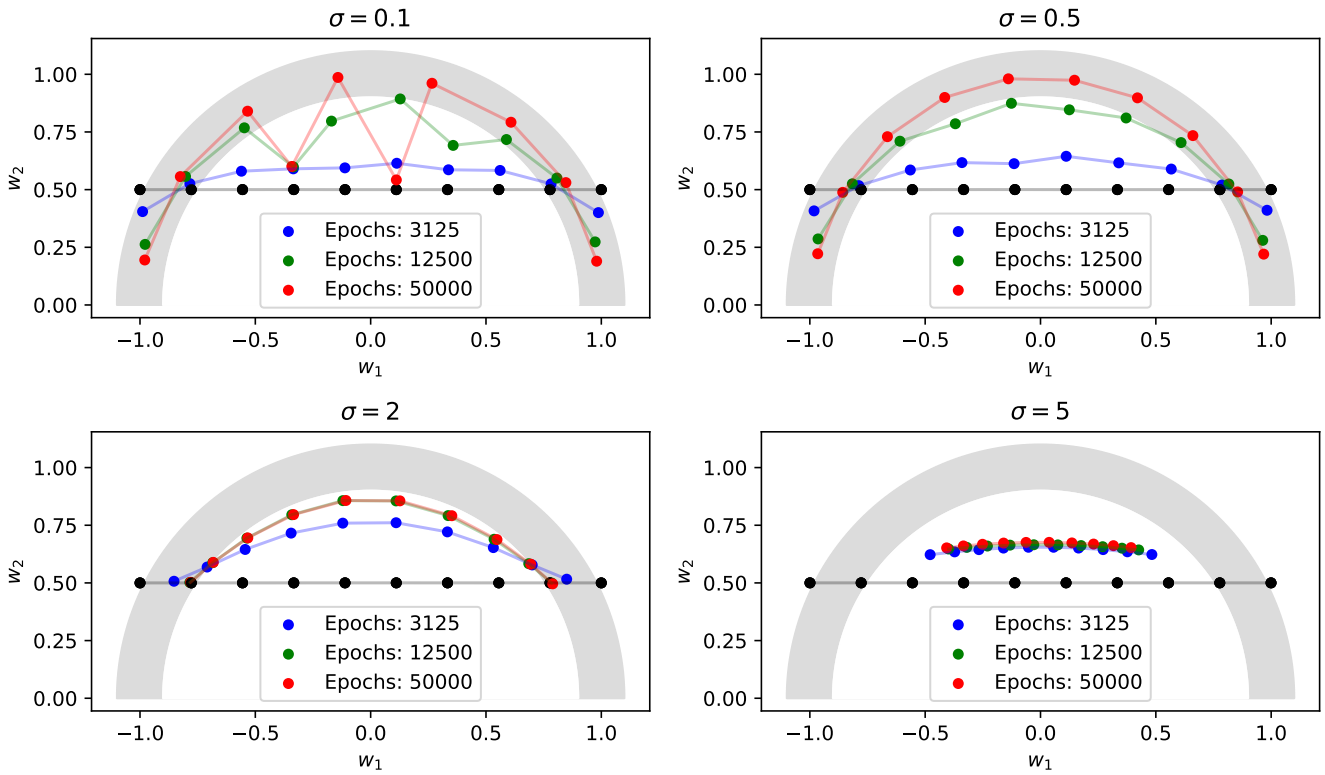


Figura 4: Resultados del entrenamiento para distintos $\sigma$ de la función de vecindad. Se ilustra los pesos luego de distintas cantidades de épocas, y la posición inicial de los pesos a modo de referencia.

# APÉNDICE

## Código Ejercicio 1

---

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable

def V(x_,w_):
    return np.dot(x_,w_)

def regla_Oja(X_,w_,eta_,epochs_):
    history_w=[w_]
    for e in range(epochs_):
        x=X_[np.random.randint(0,X_.shape[0])]
        v=V(x,w_)
        dw=eta_*v*(x-v*w_)
        w_=w_+dw
        history_w.append(w_)
    return w_, history_w

eta=1e-4
epochs=50000
sigma=np.array([[2,1,1,1],[1,2,1,1],[1,1,2,1],[1,1,1,2]])
eigenVals, eigenVecs = np.linalg.eig(sigma)
w0=np.random.uniform(-0.15,0.15,4)
X=np.random.multivariate_normal([0,0,0,0],sigma,1000000)

w, history=regla_Oja(X,w0,eta,epochs)

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(6,5))
for i in range(4):
    ax1.plot([w[i] for w in history],label='$w_'+str(i+1)+'$')
ax1.legend()
ax1.axhline(y=0.5,ls='--',color='grey',zorder=0)
ax1.text(15000,0.2,'$\lambda_{max}=$'+str(round(eigenVals.max(),0))+\
    '\n$\\vec{v}_{max}=$'+str(eigenVecs[:,1]))
ax1.set_xlabel('Epocas')
ax1.set_ylabel('$\\vec{w}$')
ax2.plot(range(epochs+1),np.linalg.norm(history,axis=1))
ax2.set_xlabel('Epocas')
ax2.set_ylabel('||$\\vec{w}$||')
ax2.axhline(y=1,ls='--',color='grey',zorder=0)

plt.setp(ax1.get_xticklabels(), visible=False)
plt.subplots_adjust(hspace=0)
plt.show()

fig, ax = plt.subplots(1, 1, figsize=(6,5))
for i in range(4):
    if i!=1:
        ax.plot([np.abs(np.dot(w,eigenVecs[:,i]))/(np.linalg.norm(w)*np.linalg.norm(eigenVecs[:,1]))
            for w in history],label='Sim($\\vec{w},\\vec{v}_'+str(i+1)+'$)')
    else:
        ax.plot([np.abs(np.dot(w,eigenVecs[:,i]))/(np.linalg.norm(w)*np.linalg.norm(eigenVecs[:,1]))
            for w in history],label='Sim($\\vec{w},\\vec{v}_{'+str(i+1)+'}$) (max.)')
ax.legend()
```

```
ax.set_xlabel('Epocas')
ax.set_ylabel('Similitud')
ax.axhline(y=1,ls='--',color='grey',zorder=0)
ax.axhline(y=0,ls='--',color='grey',zorder=0)

plt.show()
```

## Código Ejercicio 2

```
eta=1e-3
epochs=[3125,12500,50000]
colors=["blue","green","red"]
sigma=[0.1,0.5,2,5]

X=[]
while len(X)<1000000:
    x=np.random.uniform(-1.1,1.1)
    y=np.random.uniform(0,1.1)
    if np.linalg.norm([x,y])>=0.9 and np.linalg.norm([x,y])<=1.1:
        X.append(np.array([x,y]))
X=np.array(X)

fig, ax = plt.subplots(1, 1, figsize=(6,5))
ax.hist2d(X[:,0],X[:,1],bins=100,density=True)
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
ax.set_xlim(-1.1,1.1)
ax.set_ylim(0,1.1)
ax.set_aspect('equal')
ax.set_title('Distribucin de los datos de entrada')

plt.show()

def neighborFunc(i, i_, sigma_):
    return np.exp(-(((i-i_)**2)/(2*(sigma_**2))))

def nearestIndex(w,x):
    return np.argmin(np.linalg.norm(w.T-x, axis=1))

def dw(i_, w, x, sigma_):
    dW = (x - w.T)

    for col in range(len(dW)):
        Lambda = neighborFunc(col, i_, sigma_)
        dW[col] *= Lambda
    return dW.T

def kohonen_map(X_,w_,eta_,epochs_,sigma_):
    w__=np.copy(w_)
    history_w = [w__]
    for e in range(epochs_):
        x = X_[np.random.randint(0,X_.shape[0])]
        i_ = nearestIndex(w__, x)

        dW = dw(i_, w__, x, sigma_)
        w__ += eta_*dW
```

```
        history_w.append(np.copy(w__))


    return np.array(w__), np.array(history_w)

def plotMap(history_,c,sig,ax_):
    ax_.fill_between(1.1*np.cos(np.linspace(0,np.pi,10000)),\
        1.1*np.sin(np.linspace(0,np.pi,10000)),color='grey',alpha=0.1,zorder=0)
    ax_.fill_between(0.9*np.cos(np.linspace(0,np.pi,10000)),\
        0.9*np.sin(np.linspace(0,np.pi,10000)),color='white',alpha=1,zorder=0)
    ax_.scatter(history_[1,0,0],history_[1,1,0],color='black',s=20,zorder=3)
    ax_.scatter(history_[-1,0,0],history_[-1,1,0],color=c,label='Epochs:
        '+str(len(history_)-1),s=20,zorder=3)
    ax_.scatter(history_[1,0,1],history_[1,1,1],color='black',s=20,zorder=3)
    ax_.scatter(history_[-1,0,1],history_[-1,1,1],color=c,zorder=3,s=20)
    ax_.scatter(history_[1,0,2],history_[1,1,2],color='black',s=20,zorder=3)
    ax_.scatter(history_[-1,0,2],history_[-1,1,2],color=c,zorder=3,s=20)
    ax_.scatter(history_[1,0,3],history_[1,1,3],color='black',s=20,zorder=3)
    ax_.scatter(history_[-1,0,3],history_[-1,1,3],color=c,zorder=3,s=20)
    ax_.scatter(history_[1,0,4],history_[1,1,4],color='black',s=20,zorder=3)
    ax_.scatter(history_[-1,0,4],history_[-1,1,4],color=c,zorder=3,s=20)
    ax_.scatter(history_[1,0,5],history_[1,1,5],color='black',s=20,zorder=3)
    ax_.scatter(history_[-1,0,5],history_[-1,1,5],color=c,zorder=3,s=20)
    ax_.scatter(history_[1,0,6],history_[1,1,6],color='black',s=20,zorder=3)
    ax_.scatter(history_[-1,0,6],history_[-1,1,6],color=c,zorder=3,s=20)
    ax_.scatter(history_[1,0,7],history_[1,1,7],color='black',s=20,zorder=3)
    ax_.scatter(history_[-1,0,7],history_[-1,1,7],color=c,zorder=3,s=20)
    ax_.scatter(history_[1,0,8],history_[1,1,8],color='black',s=20,zorder=3)
    ax_.scatter(history_[-1,0,8],history_[-1,1,8],color=c,zorder=3,s=20)
    ax_.scatter(history_[1,0,9],history_[1,1,9],color='black',s=20,zorder=3)
    ax_.scatter(history_[-1,0,9],history_[-1,1,9],color=c,zorder=3,s=20)

    ax_.plot(history_[1,0,:],history_[1,1,:],zorder=3,color="black",alpha=.1)
    ax_.plot(history_[-1,0,:],history_[-1,1,:],zorder=3,color=c,alpha=.3)

    ax_.set_xlabel('$w_1$')
    ax_.set_ylabel('$w_2$')
    ax_.legend(loc="lower center")
    ax_.set_title('$\sigma='+str(sig)+'$')

fig, ax = plt.subplots(2,2,figsize=(11,6.5))
axs = [ax[0,0],ax[0,1],ax[1,0],ax[1,1]]
cont=0
for j in range(len(sigma)):
    axs[cont].set_aspect('equal')
    for i in range(len(epochs)):
        w0 = np.array([np.linspace(-1,1,10),np.array([0.5 for i in range(10)])])
        w, history_w = kohonen_map(X,w0,eta,epochs[i],sigma[j])

        plotMap(history_w,colors[i],sigma[j],axs[cont])
    cont+=1

plt.show()
```