APLICACIÓN DE RED NEURONAL SOM EN PROGRAMACIÓN PARALELA

MG. HUAROTE ZEGARRA RAÚL.

OBJETIVOS:

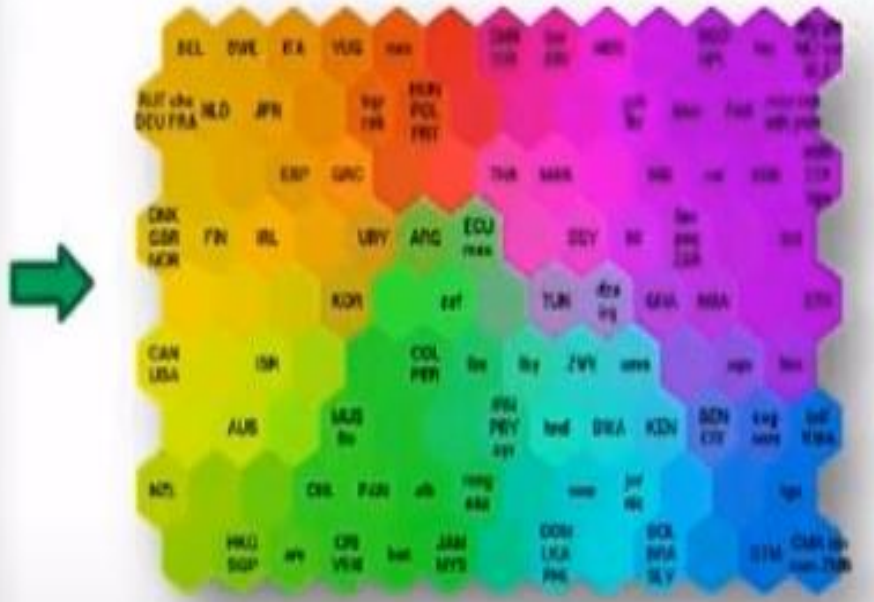
- Conoce los fundamentos de la redes neuronales SOM.
- Aplicar la programación paralela en los proceso de aprendizaje y comprobación

TEMAS A TRATAR:

- Redes no supervisadas competitivas.
- Aplicaciones.
- Arquitectura de la red.
- Entrenamiento de la red.

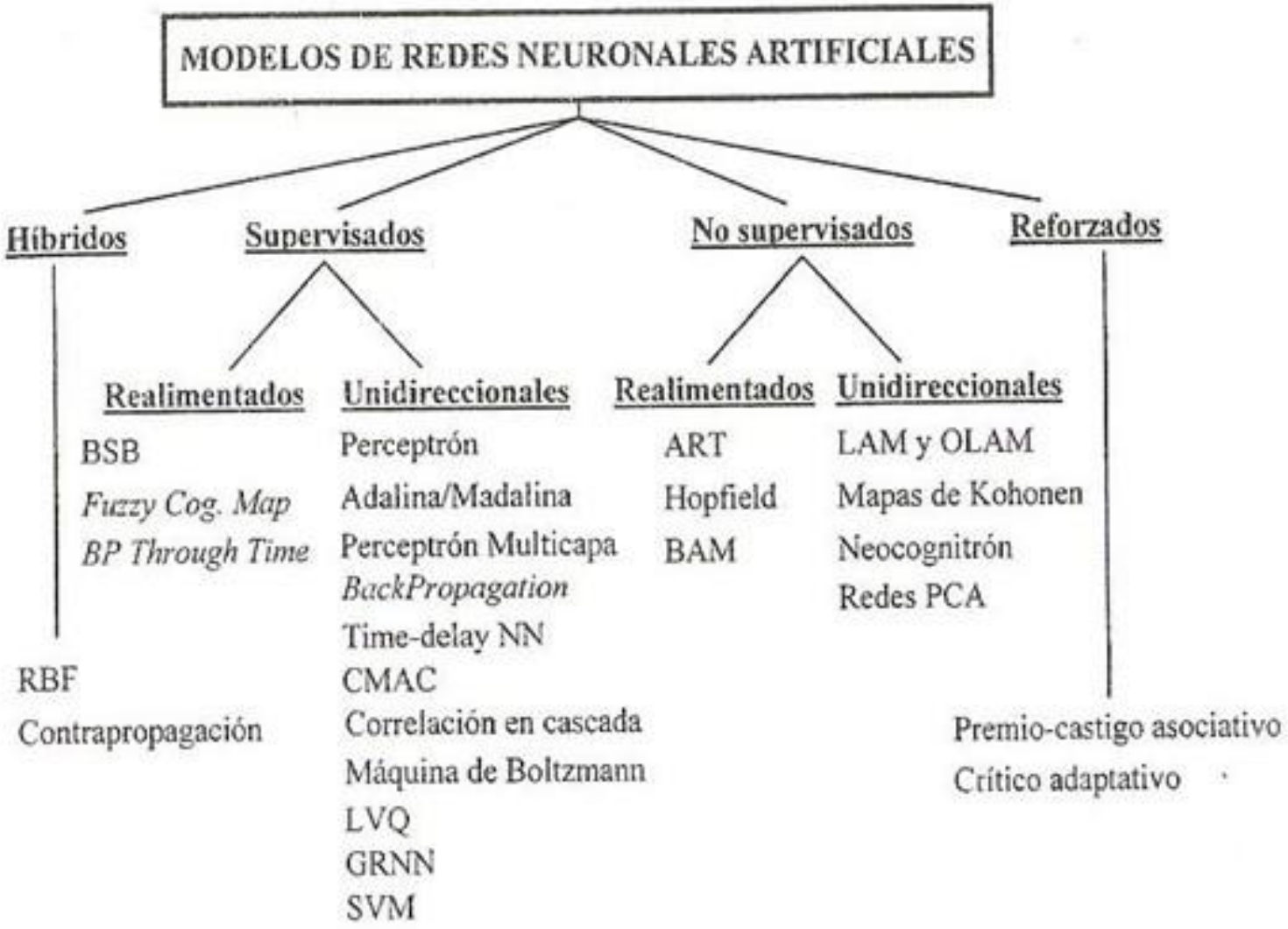
MOTIVACIÓN:

	A	B	C	D	E
1	Country	Country C	Health Ex	Education E	Inflation
2	Aruba	ABW	9.418971	5.92467022	-2.13637
3	Afghanist	AFG	4.371774		-8.28308
4	Angola	AGO	5.791339		13.73145
5	Albania	ALB	6.75969		2.280502
6	Andorra	AND	4.57058	3.1638701	
7	Arab Wor	ARB	4.049924		3.524814
8	United Ar	ARE	7.634758		
9	Argentina	ARG	4.545323	4.88997984	6.282774
10	Armenia	ARM		3.84079003	3.406767
11	American	ASM	4.862062		
12	Antigua a	ATG	9.046056	2.55447006	-0.55016
13	Australia	AUS	11.19444	5.09262991	1.820112
14	Austria	AUT	5.85024	5.7674098	0.506313
15	Azerbaija	AZE	6.964187	3.22430992	1.401056
16	Burundi	BDI	10.39434	6.3197999	10.98147
17	Belgium	BEL	4.46431	6.41535997	-0.05315
18	Benin	BEN	7.405431	4.22204018	2.15683



ENTORNO DE LAS REDES NEURONALES:

Ubicación de la red SOM



RED NEURONAL SOM:

Caracterizado por que en el proceso de entrenamiento no tiene objetivo o salidas deseadas.

Aplicado principalmente en agrupamientos de patrones característicos o clúster, análisis exploratorio de datos y en minería de datos.

Modifica su red completa para llevar a cabo un objetivo específico.

Las neuronas compiten con otras para llevar a cabo una tarea dada.

RED NEURONAL SOM: TEUVO KOHONEN

Es un científico de la computación, informatólogo y académico finlandés. Actualmente es profesor emérito de la Academia de Finlandia. Professor of Computer Science, Computer Science Department, Biology Department, Department for Physics and Astronomy. Director, Laboratory of Computational and Biological Vision Director, Institut für Neuroinformatik , Ruhr-Universität Bochum, Germany Ph D. Christoph von der Malsburg (1970's).
Creó los Self-organizing Feature Map (SOFM) que significa "Red de Mapas Auto-organizativos.

Teuvo Kohonen / Libros

RED NEURONAL SOM:

Redes no supervisadas competitivas

Solamente una neurona (o grupo de vecinas) puede quedar finalmente activadas. La base de la operación de estos modelos es la competencia entre las neuronas, materializada en inhibiciones laterales.

Durante la fase de aprendizaje las neuronas ganadoras obtienen como premio el refuerzo de sus conexiones sinápticas.

Comportamiento básico en muchos de los modelos auto-organizados mas conocidos (ART, Neo-cognitron y Mapas Auto-organizados de Kohonen).

RED NEURONAL SOM:

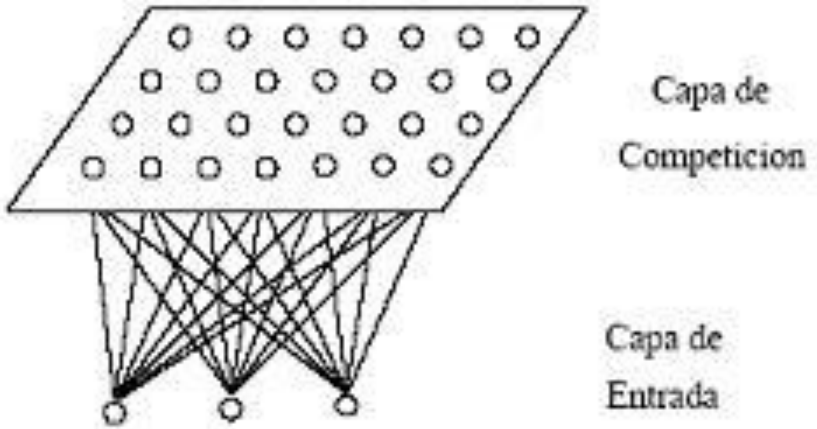
Aplicaciones:

- Clasificación de patrones.
- Cuantificación vectorial.
- Reducción de dimensiones.
- Extracción de rasgos.
- Monitorización de procesos.
- Análisis exploratorio.
- Visualización.
- Minería de datos.
- Etc.

RED NEURONAL SOM:

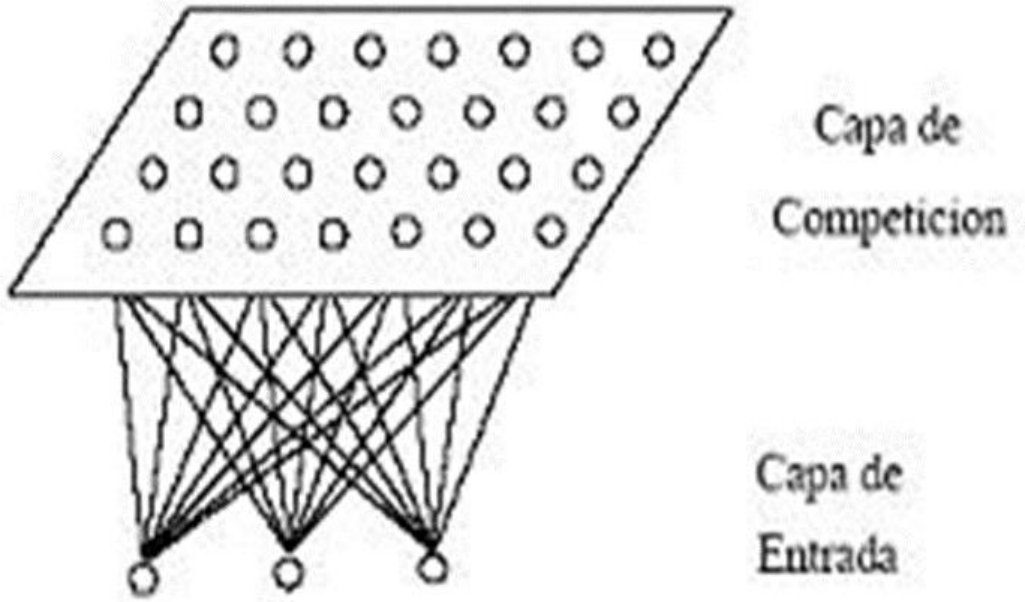
Los mapas auto-organizados de Kohonen o SOFM (Self-Organizing Feature Maps) intentan modelar, de forma sencilla, las redes neuronales naturales.

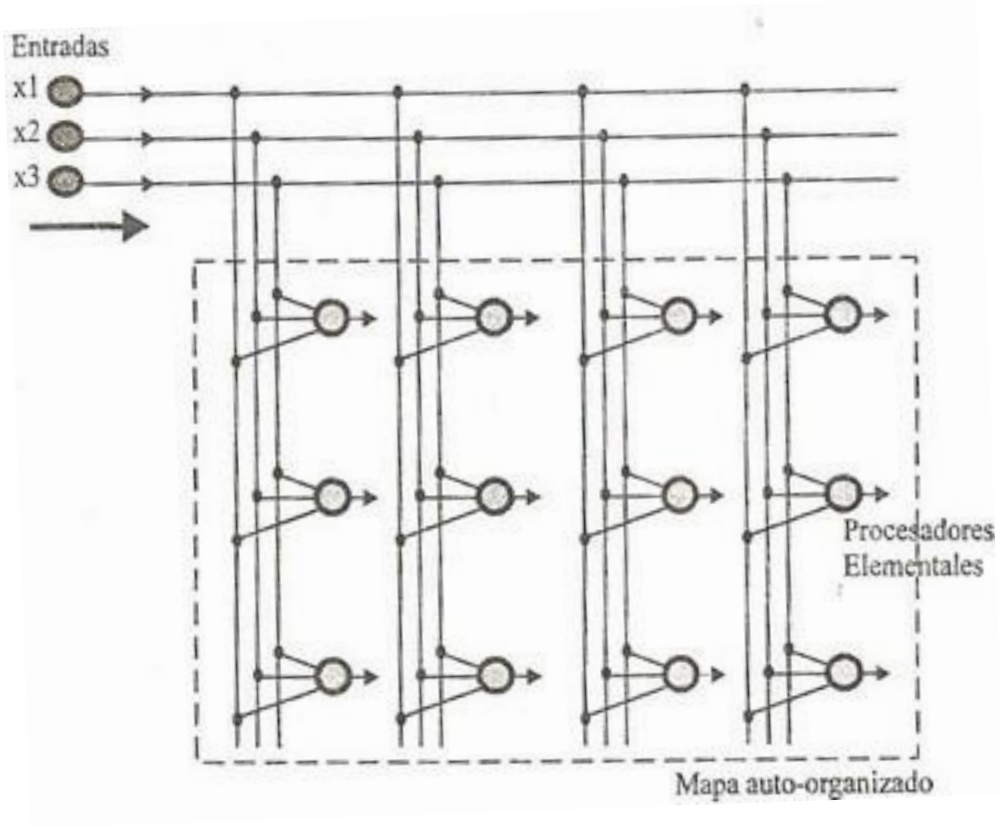
En este modelo las neuronas se organizan en una arquitectura unidireccional de dos capas.
La primera es la **capa de entrada** o sensorial, que consiste en *m* neuronas, una por cada variable de entrada. Las entradas $\mathrm{x(t)} \in R^m$ son muestras del espacio sensorial.
La segunda es la **capa de competición** o procesamiento, consiste habitualmente en una estructura rectangular $\mathrm{n(x)\,X\,n(y)}$ neuronas que **operan en paralelo**.



RED NEURONAL SOM:

Arquitectura de la red:





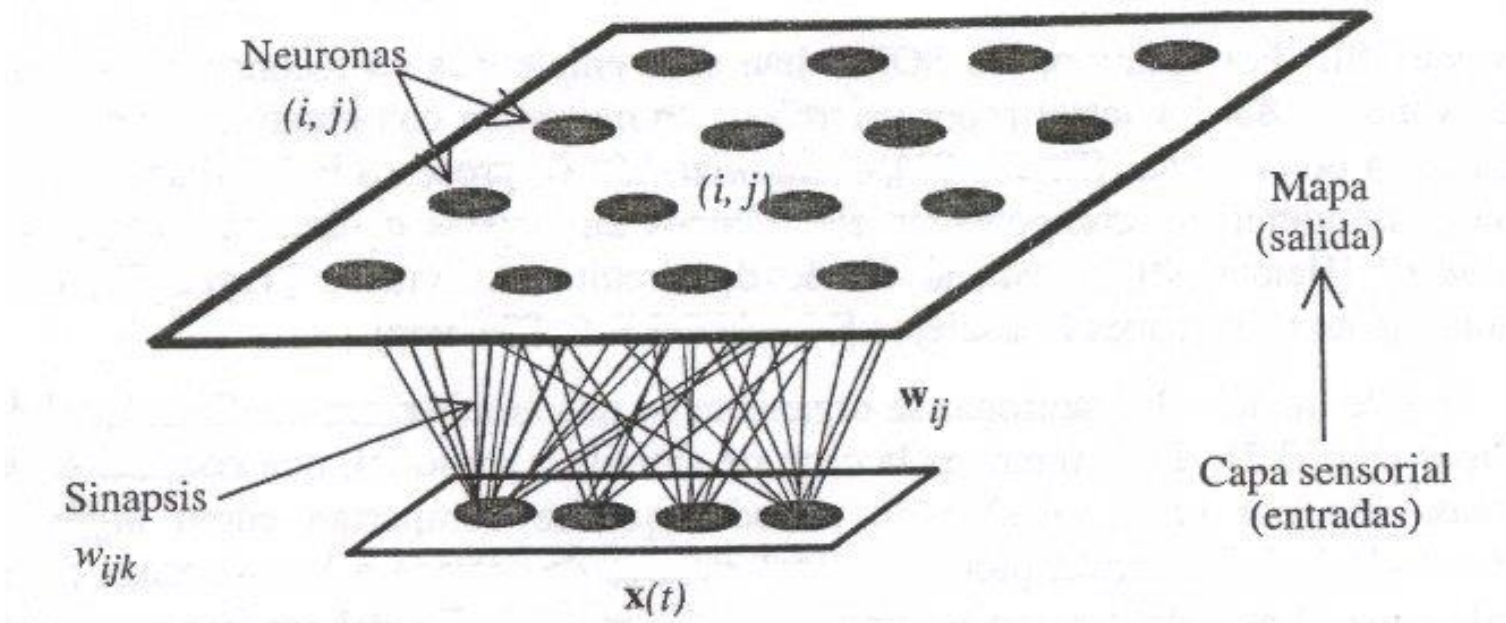
RED NEURONAL SOM:

Fase de ejecución (operación normal de la red)

Cada neurona (i, j) calcula la similitud entre el vector de entrada X y su propio vector de pesos sinápticos, según una cierta medida de distancia o criterio de similitud.

Se declara vencedora la neurona, cuyo vector de pesos sinápticos es similar al vector de entrada.

$$\min_{ij}\{d(\mathbf{w}_{ij}, \mathbf{x})\}$$



RED NEURONAL SOM:

Fase de aprendizaje

Se presenta un vector de entradas **X(t)**.

La neurona vencedora modifica sus pesos de tal manera que se parezca mas a **X(t).**

El proceso se repite para numerosos patrones de entrada, de forma que al final se tiende a representar la función de probabilidad p(X) (o función de distribución) del espacio sensorial.

Si el espacio sensorial esta dividido en grupos, cada neurona se especializa en uno de ellos, y la operación esencial de la red se pude interpretar entonces como un análisis clúster.

RED NEURONAL SOM:

Fase de aprendizaje

Este modelo también incorpora al esquema relaciones entre neuronas próximas en el mapa. Para ello introduce una función de vecindad.

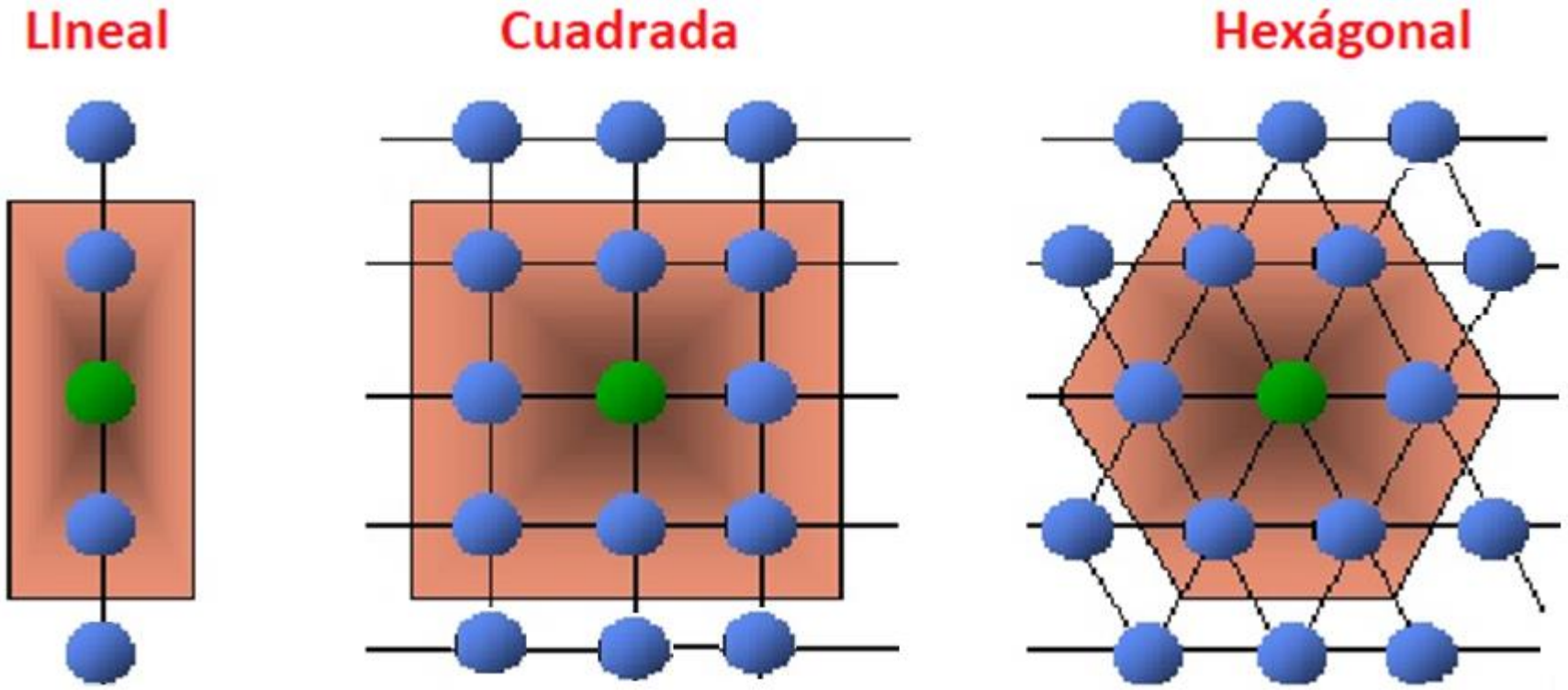
El efecto de la función de vecindad es que durante el proceso de aprendizaje se actualizan tanto los pesos de la neurona vencedora como los de las neuronas pertenecientes a su entorno.

De esta manera se refleja sobre el mapa una cierta imagen del orden topológico presente en el espacio de entrada (ordenación de los detectores de rasgos).

RED NEURONAL SOM:

Vecindad

Indica que nodos aprenden cuando se activa una neurona ganadora.



Teuvo Kohonen sugiere usar las cuadradas y hexagonales para obtener un resultado eficiente.

RED NEURONAL SOM:

Un algoritmo de aprendizaje autoorganizado habitual

1. *Inicialización de pesos sinápticos* W_{ijk}: en t=0 se puede partir de pesos nulos, aleatorios pequeños (lo mas habitual) o con pesos de partida predeterminados.

Para cada iteración

2. *Presentación de patrones X(t)*: cuando disponemos de un conjunto finito de patrones de entrenamiento (lo mas habitual), basta con tomar al azar un patrón y presentarlo a la red.

RED NEURONAL SOM:

3. *Calcular similitud*: cada neurona i = (i, j) calcula la similitud entre el vectores de entradas y el vector de pesos sinápticos. Un criterio de medida muy utilizado es la distancia euclidiana:

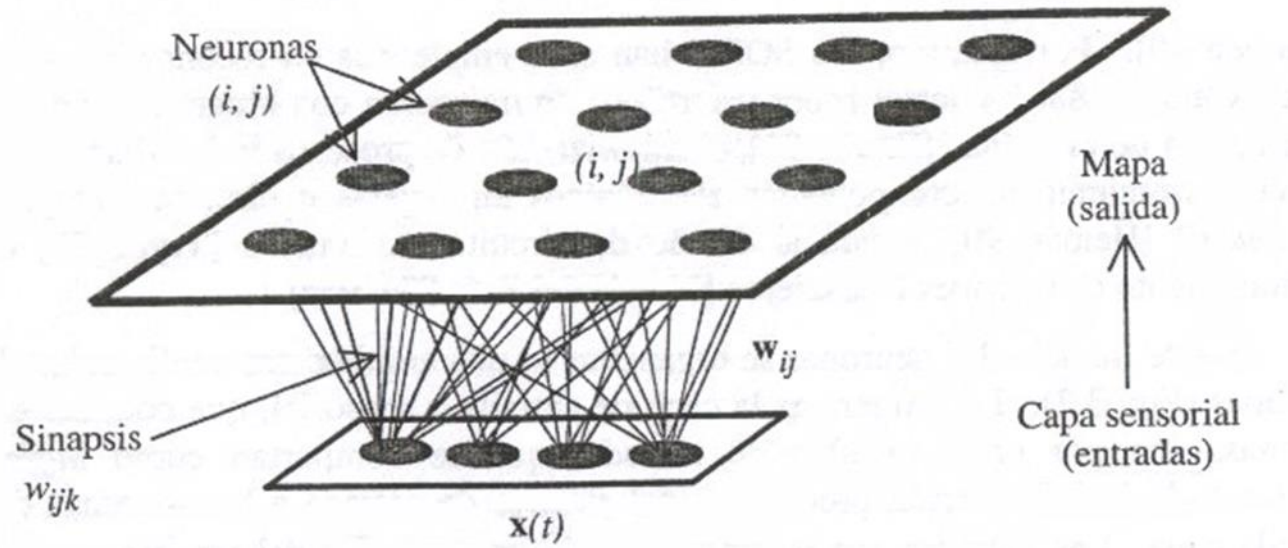
$$d^2(\mathbf{w}_{ij}, \mathbf{x}) = \sum_{k=1}^{n} \left(w_{ijk} - x_k \right)^2$$

4. *Determinación de la neurona ganadora*: se selecciona la neurona, cuya sum(e.*w')/sqrt(sum(e.^2)*sum(w.^2)) sea la menor de todos.

RED NEURONAL SOM:

5. *Actualización de los pesos sinápticos* de la neurona ganadora y los de las neuronas vecinas: la regla de aprendizaje mas empleadas es:

$$w_{ijk}(t+1) = w_{ijk}(t) + \alpha(t).h(|\mathbf{i}-\mathbf{g}|,t).\left(x_k(t) - w_{ijk}(t)\right)$$



RED NEURONAL SOM:

$W_{ijk}(t)$: Pesos sináptico entre la neurona de entrada k y la neurona de salida (i, j), en la iteración actual.

$W_{ijk}(t+1)$: Pesos sináptico entre la neurona de entrada k y la neurona de salida (i, j), para la siguiente iteración.

$\alpha(t)$: Ritmo de aprendizaje.

h(.): función de vecindad. Es la encargada de definir que neuronas son las vecinas de la actualmente ganadora. Esta función depende de la distancia de la neurona i=(i, j) y la ganadora g=(g1, g2) , valiendo cero cuando la neurona **i** no pertenece a la vecindad de **g** (con lo que sus pesos no son actualizados, y un numero positivo cuando si pertenece (sus pesos si son modificados).

$$|\mathbf{i}-\mathbf{g}|=\sqrt{(i-g1)^2+(j-g2)^2}$$

RED NEURONAL SOM:

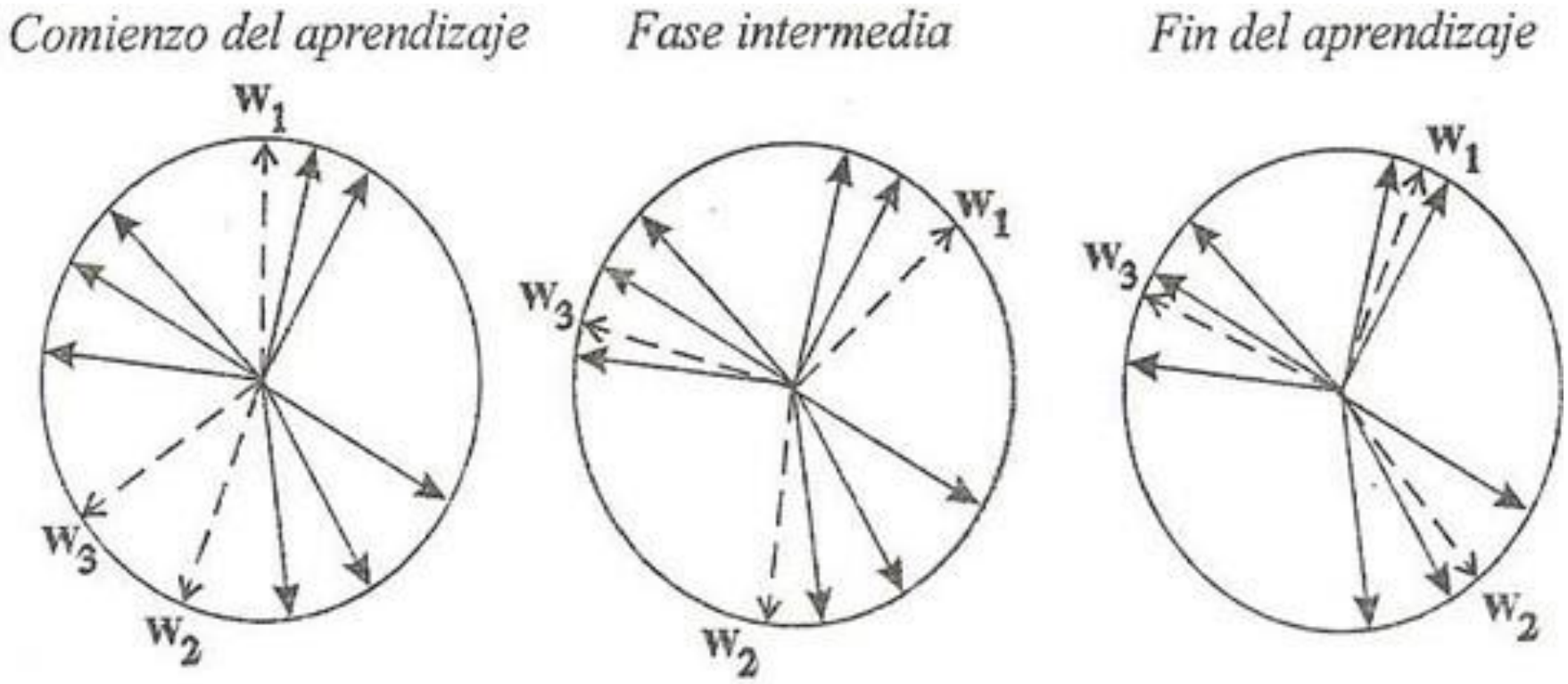
6. Si se ha alcanzado el **numero máximo de iteraciones** establecido, entonces el proceso de aprendizaje finaliza. En caso contrario se vuelve al paso 2.

El numero de iteraciones debe ser suficientemente grande, así como proporcional al numero de neuronas del mapa (a mas neuronas, es necesario mas iteraciones), e independiente del número de componentes de **X**. *Aunque 500 iteraciones por neurona es una cifra adecuada*, de 50 a 100 suelen ser suficientes para la mayor parte de los problemas.

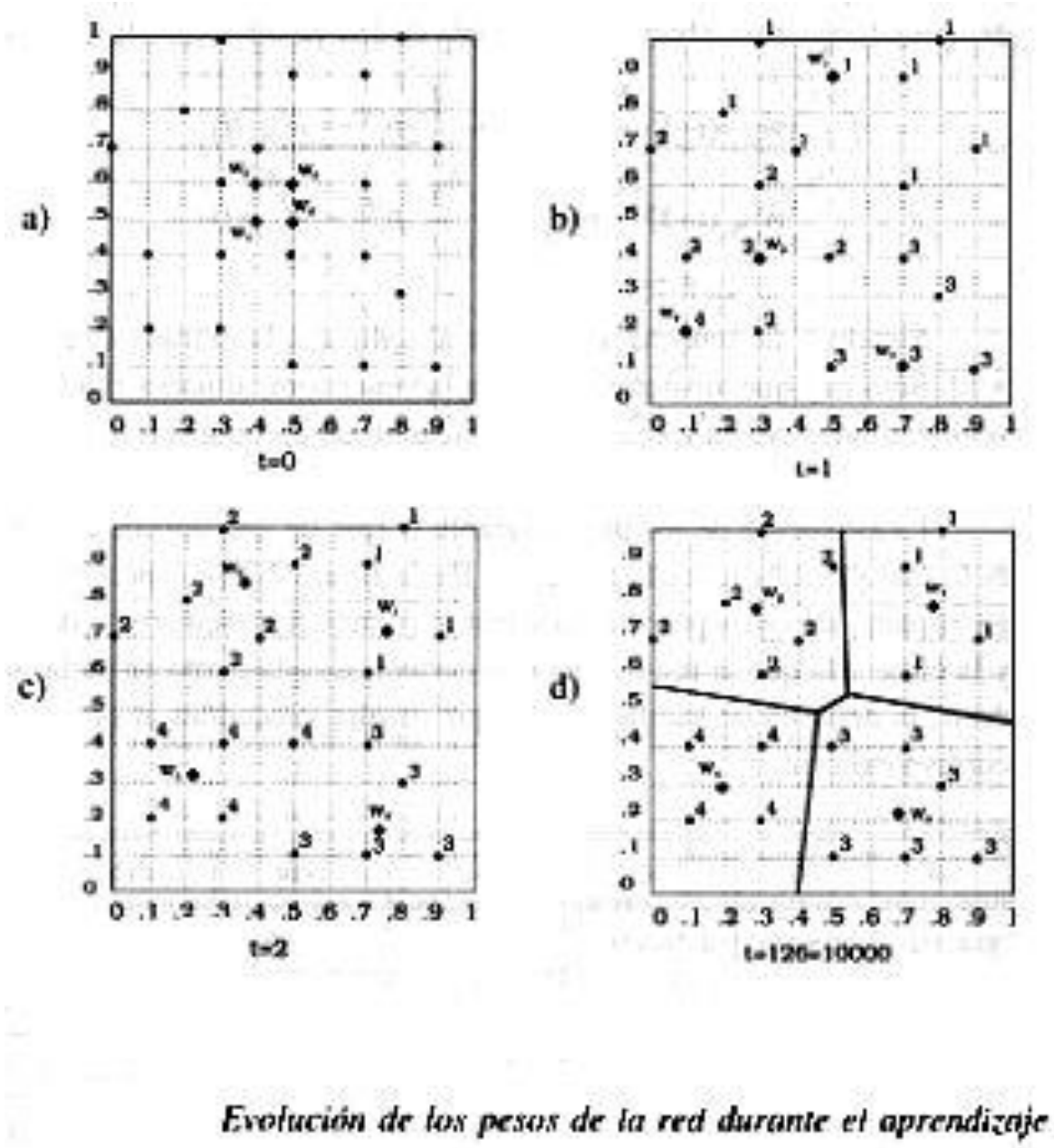
RED NEURONAL SOM:

Interpretación del algoritmo de aprendizaje

Operación de la misma regla de aprendizaje para el caso de varios patrones pertenecientes al espacio de entrada, representados por vectores de trazo continuo.



RED NEURONAL SOM:

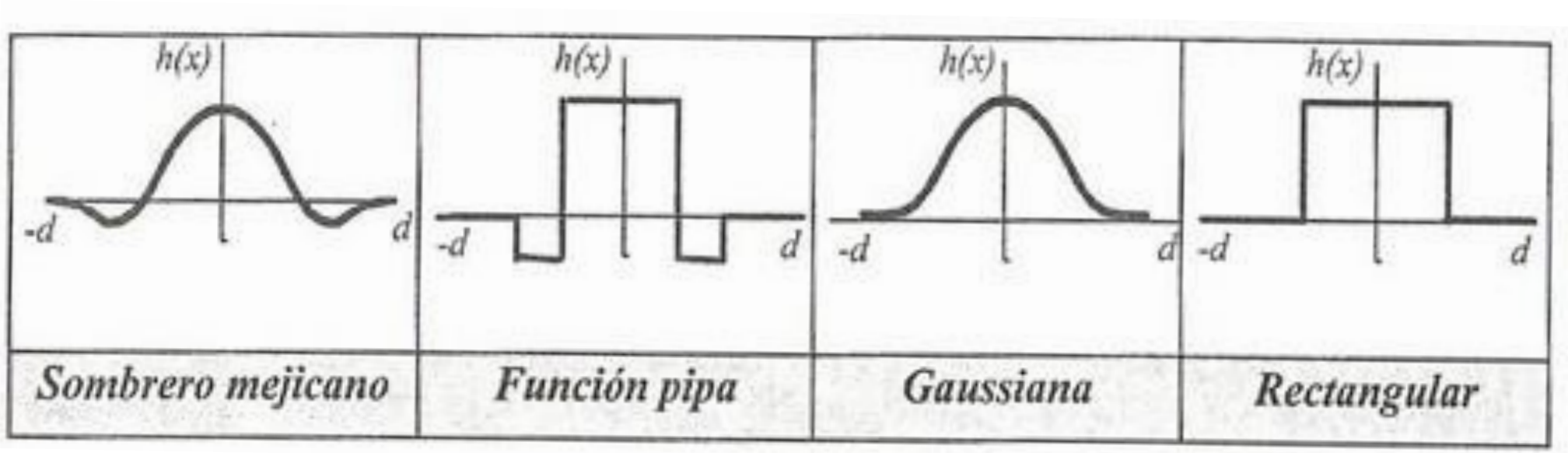


Evolución de los pesos de la red durante el aprendizaje

RED NEURONAL SOM:

Función de vecindad

Existen diferentes formas de la función vecindad. En general h(.) decrece con la distancia de la vencedora.

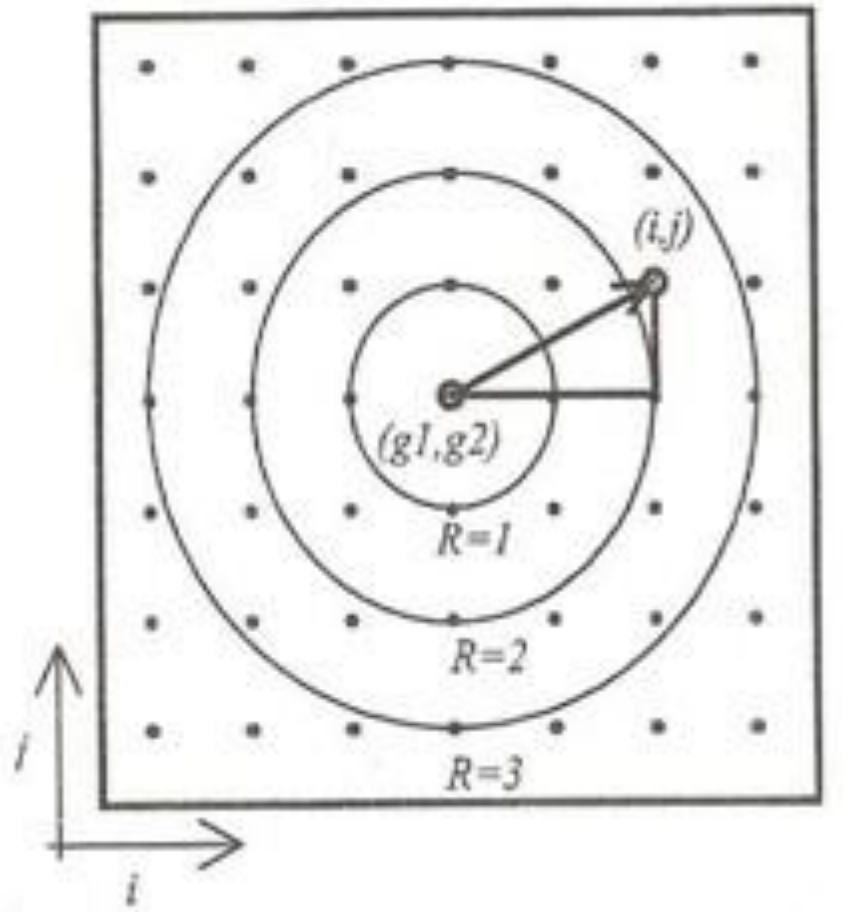


RED NEURONAL SOM:

La función de vecindad mas simple es del **tipo escalón**, que denominaremos rectangular.

$$h(|\mathbf{i}-\mathbf{g}|,t)=\begin{cases}0, & \text{si } |\mathbf{i}-\mathbf{g}|>R(t)\\ 1, & \text{si } |\mathbf{i}-\mathbf{g}|\leq R(t)\end{cases}$$

Si utilizamos la función de vecindad rectangular, una neurona i pertenece a la vecindad de la ganadora solamente si su vecindad es inferior a R(t). Con este tipo de función la vecindad adquiere forma circular de bordes nítidos, en torno a la vencedora.



RED NEURONAL SOM:

- Si utilizamos la función **de vecindad rectangular** la regla de aprendizaje se reduce a:

$$\Delta w_{ijk}(t) = \begin{cases} 0, & \text{si } |\mathbf{i}-\mathbf{g}| > R(t) \\ \alpha(t)\left(x_k(t) - w_{ijk}(t)\right), & \text{si } |\mathbf{i}-\mathbf{g}| \leq R(t) \end{cases}$$

  Por lo que en cada iteración solo se actualizan las neuronas que distan de la vencedoras menos de R(t).

- Si utilizamos funciones de **vecindad gaussianas** o en forma de sombrero mexicano, establecemos niveles de pertenencia en lugar de fronteras nítidas.

$$w_{ijk}(t+1) = w_{ijk}(t) + \alpha(t).h(|\mathbf{i}-\mathbf{g}|,t).\left(x_k(t) - w_{ijk}(t)\right)$$

RED NEURONAL SOM:

donde α_f y α_i corresponden a las tasas de aprendizaje final e inicial respectivamente; t_{max} es el número máximo de iteraciones.

$$\propto(t)=\propto_i=\left(\frac{\propto_f}{\propto_i}\right)^{\frac{1}{t_{max}}}$$

Con esta expresión lo que se busca es que la tasa de aprendizaje siga una función exponencial con el fin de tener al inicio del proceso fuertes variaciones en los pesos y, a medida que avance el proceso, las variaciones disminuyan para así garantizar que al inicio del proceso las neuronas se distribuyan lo más rápido posible entre los datos representativos de la base de entrenamiento.

RED NEURONAL SOM:

$$\alpha(n) = \frac{1}{n} \qquad \alpha(n) = \alpha_1\left(1 - \frac{n}{\alpha_2}\right)$$

Siendo α_1 un valor de 0.1 ó 0.2 y α_2 un valor próximo al número total de iteraciones del aprendizaje. Suele tomarse un valor $\alpha_2 = 10000$.

RED NEURONAL SOM:

Algoritmo de entrenamiento

Algorithm 1 Estructura del algoritmo de entrenamiento de la red SOM

```
1: Inicialización de parámetros
2: repeat
3:     Presentar el vector de entrada.
4:     Buscar neurona ganadora.
5:     Actualizar los pesos.
6:     Modificar la tasa de aprendizaje.
7: until Se cumple criterio de parada
```

CASO PRACTICO:

APLICACIÓN DE LA RED NEURONAL SOM:

Clasificación de Tipos de Huella Dactilar con Red Neuronal SOM

RESUMEN:

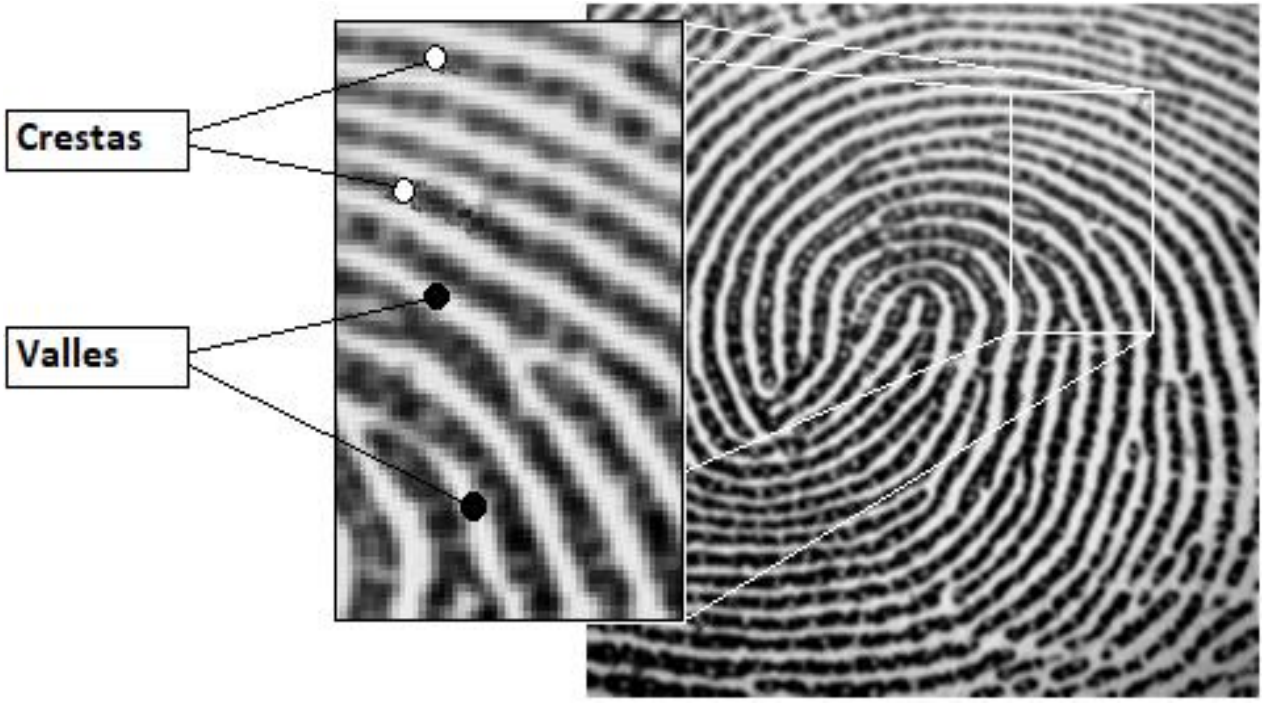
Clasificar las huellas dactilares usando la red neuronal artificial de tipo mapa auto-organizativo (self organizing map, SOM) de Teuvo Kohonen, y como medio de aprendizaje de la red usar las orientaciones de las líneas marcadas por las crestas y valles, a pesar de haber ruido presente.

Para el proceso de aprendizaje usar imágenes de huellas de diferentes tipos (ARCO, LEFT LOOP, RIGHT LOOP y WHORL). Para la comprobación usar imágenes, con ruido de tipo ‘salt and pepper’ con densidad 0.03.

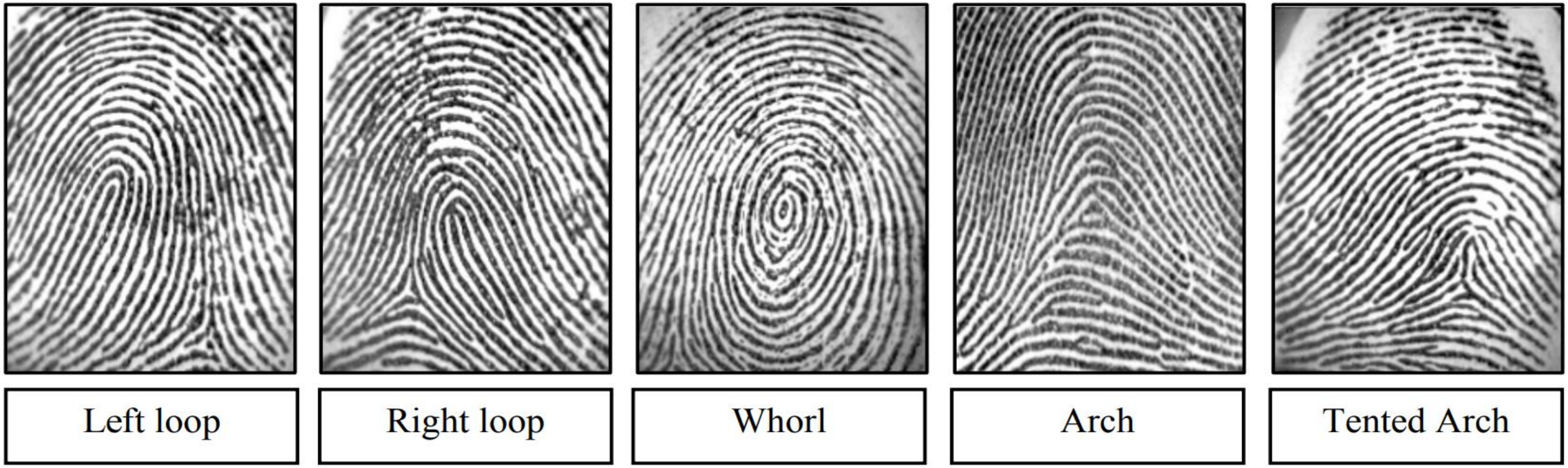
HUELLA DACTILAR:

- “Es un ser individual” **C. T. Guillermo (2006)**
- “Medio de individualización y una herramienta forense de valor incalculable” **G. Francis (1892)**
- “La identificación segura de la persona es por medio de su huella dactilar” **L. Hong, Y. Wang, A. K. Jain (1998).**

Fuente: **G. Francis (1892)**



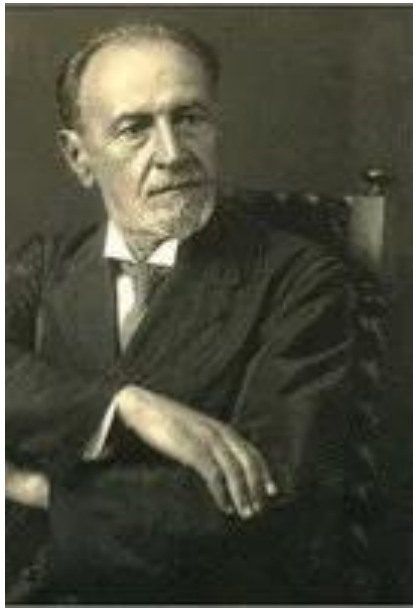
TIPOS DE HUELLA DACTILAR:



Fuente: **M. Davide, M. Dario, J. Anil, P. Salil (2009)**

PRIMER CASO HISTÓRICO:

Desde que el Croata nacionalizado Argentino **Juan Vucetich** (1951) desarrolló y puso por primera vez en práctica un sistema eficaz de identificación de personas por medio de sus huellas dactilares, puesto que por primera vez en el mundo hizo un registro dactiloscópico de las personas y por medio de ello en 1982 se identificó y se condenó a una asesina llamada *Francisca Rojas de Caraballo*, en base a sus huellas ensangrentadas dejadas en la escena del crimen.



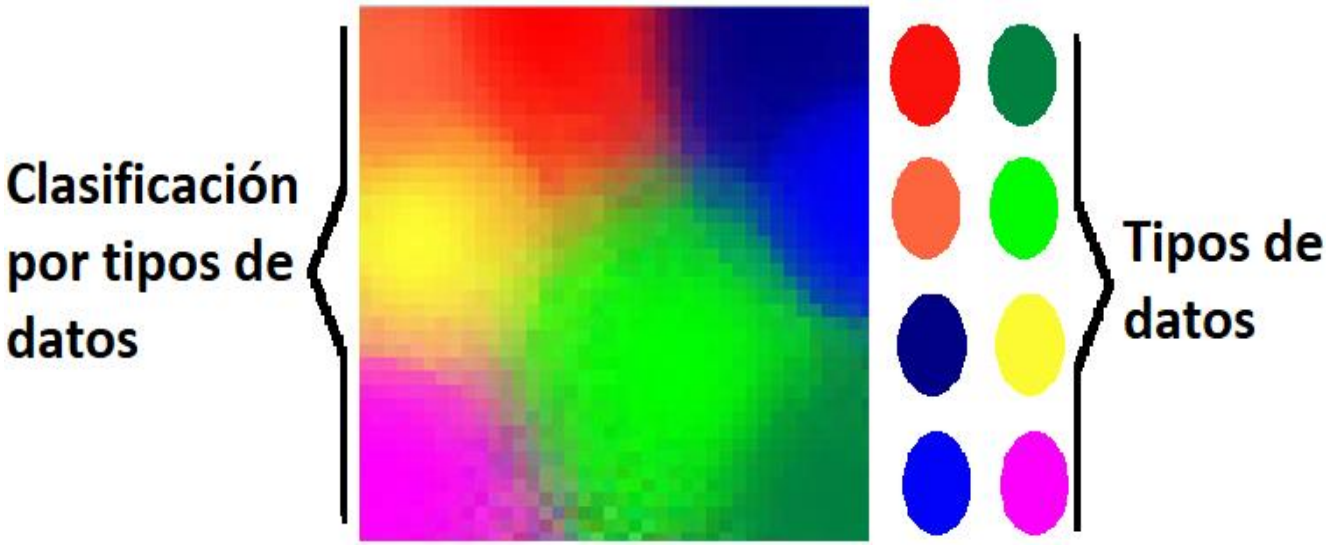
Fowkes Rhodes (1956) remarca que Juan Vucentich marcó un antes y un después respecto a la identificación de personas por medio de las huellas dactilares.

TIPOS DE RED NEURONAL ARTIFICIAL:

Tipos de Redes neuronales	Detalles
Retropropagación	Nace con (Werbos,1974) desarrollando la idea básica del algoritmo de aprendizaje de propagación hacia atraz. Posteriormente redescubierta por (Rumelhart & Ginton & Williams, 1986) demostrando que puede resolver problemas de tipo no-lineales, lo cual representa a los problemas del mundo real.
Mapa Autoorganizativo	La idea de (Kohonen, 2001) es crear una red neuronal adaptativo y que pueda autoorganizar la información que recibe mediante una etapa de aprendizaje, donde se modifica la red neuronal completa

RED NEURONAL SOM:

“Toma este modelo biológico para clasificar de acuerdo a características comunes, este tipo de aprendizaje no se considera un tutor que le esté indicando si la red neuronal esta operando correctamente o no, ya que no dispone de un valor deseado en el cual deba tender” **Teuvo Kohonen (1989, 1990).**



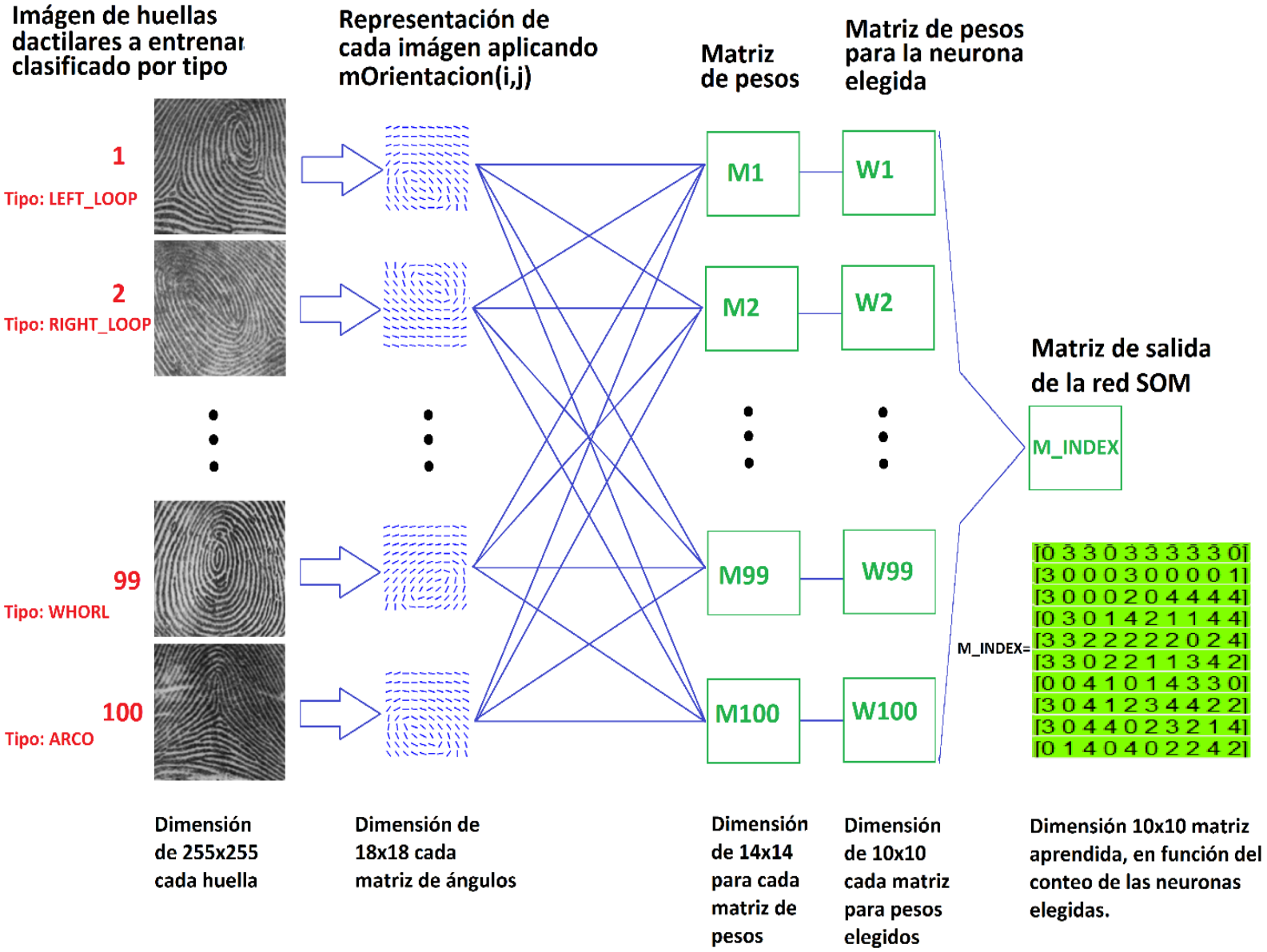
$$\min \left\| X_p - W_j \right\| = \max \sum_{i=1}^{N} x_{pi} \cdot w_{ji}$$

$$w_{ji}(t+1) = w_{ji}(t) + ß(t)\,[e_i^{(k)} - w_{j*i}(t)]$$

para j perteneciente a $Zona_{j*}(t)$

$$ß(t) = 1/t$$

ESTRATEGIA PARA CLASIFICAR TIPO DE HUELLA



Las huellas dactilares de dimensión 256x256 sea representado en una matriz de orientaciones, teniendo como resultado por cada huella una matriz de orientaciones 'E' de dimensión 18x18, y esta matriz de orientaciones vienen a ser los patrones de entrada a la red neuronal. En el proceso de entrenamiento la matriz de pesos 'M' son generados de manera aleatoria y tienen dimensión de 14x14, para mejor simplicidad se crea una matriz 'W' de dimensión 10x10 donde va a almacenar las neuronas elegidas y sus vecinas, el conjunto de neuronas elegidas se extrae en una matriz de salidas M_INDEX, donde se contabiliza las veces que ha sido elegido y por tanto se puede almacenar la clase de tipos de huellas dactilares.

ESTRATEGIA PARA CLASIFICAR TIPO DE HUELLA

Parámetros para el proceso de entrenamiento de la RED SOM:

- **Factor de aprendizaje: Alpha_max = 0,45.** Este es el factor de aprendizaje inicial, que va a ir decrecerá con el tiempo.

- **Vecindad Gaussiana:**

```
vecindad_gaussiana= [0       0       0.125   0       0;
                     0       0.25    0.5     0.25    0;
                     0.125   0.5     1       0.5     0.125;
                     0       0.25    0.5     0.25    0;
                     0       0       0.125   0       0 ];
```

- **Máximo numero de iteraciones:** 500 iteraciones

- **Clases de Huellas Dactilares**

```
%clases de huellas dactilares
global LEFT_LOOP;
global RIGHT_LOOP;
global WHORL;
global ARCO;

LEFT_LOOP   = 1;
RIGHT_LOOP  = 2;
WHORL       = 3;
ARCO        = 4;
```

CONCLUSIONES:

Con el uso de la red neuronal SOM para la clasificación tipos de huella dactilar, se logra los siguientes resultados:

- Se demuestra que se puede extraer patrones carateristicos de una huella a partir de los ángulos entre las crestas y valles por medio del filtro de orientación, y que estos patrones obtenidos son entrada a la red SOM.
- Se logra demostrar una tasa de efectividad de 96.25% con ruido de tipo sal y pimienta con densidad 0.03.
- Para el proceso de entrenamiento de las huellas, se realiza con 1000 iteraciones, alpha máximo de 0.45 (esto se reduce en el proceso) y de dimensión de cada imagen de la huella de 256x256 pixeles en formato .tif.

RECOMENDACIONES:

- Para futuro tratamiento o investigación como la búsqueda rápida del individuo, aporte a publicaciones como Balder (1991) “Dermatoglypics” en la proporción de tipos de huellas dactilares en una determinada región, país, etc.
- Puede realizar futuras investigación realizando la evaluación de diferentes tipos de ruidos, como el gaussiano, speckle, etc.
- Matworks (2018 y posterior) muestra que tiene bloques propios de la red neuronal SOM, pero se ha considerado necesario realizarlo a mano para el ingreso de orientaciones, y poder usarlo en clasificaciones de tipos de huellas dactilares e identificación de cada una de las imágenes de huella.

CODIGO FUENTE DE LA RED SOM (SECUENCIAL)

kohonenhuella.py

```python
import numpy as np
import math
from PIL import Image
# datos globales
LEFT_LOOP    = 1
RIGHT_LOOP   = 2
WHORL        = 3
ARCO         = 4
vecindad_gaussiana = np.array(
    [[0,0,0.125,0,0],
     [0,0.25,0.5,0.25,0],
     [0.125,0.5,1,0.5,0.125],
     [0,0.25,0.5,0.25,0],
     [0,0,0.125,0,0]]
    ,np.float32)

# crear una matriz de pesos sinapticos
W = np.zeros([324,14,14])
for i in range(324):
    W[i,2:12,2:12] = np.random.rand(10,10)
# Crear una matriz temporal
M_t = np.zeros([14,14],np.float32)
# crear la matriz de indices o resultante de lo aprendido
```

CODIGO FUENTE DE LA RED SOM (SECUENCIAL)

kohonenhuella.py

```
# crear la matriz de indices o resultante de lo aprendido
M_index = np.zeros([10,10],np.float32)
max_iter = 500# maximo cantidad de generaciones
alpha_max = 0.45#factor de aprendizaje inicial

def sobel(patron):
    m,n = patron.shape
    I = np.array(patron,np.float32)
    Gx = np.zeros([m-2,n-2],np.float32)
    Gy = np.zeros([m-2,n-2],np.float32)
    gx = [[-1,0,1],[-2,0,2],[-1,0,1]]
    gy = [[1,2,1],[0,0,0],[-1,-2,-1]]
    for i in range(1,m-2):
        for j in range(1,n-2):
            Gx[i-1,j-1] = sum(sum(I[i-1:i+2,j-1:j+2]*gx))
            Gy[i-1,j-1] = sum(sum(I[i-1:i+2,j-1:j+2]*gy))
    return Gx,Gy
def medfilt2(G,d=3):
    m,n = G.shape
    temp = np.zeros([m+2*(d//2),n+2*(d//2)],np.float32)
    salida = np.zeros([m,n],np.float32)
    temp[1:m+1,1:n+1] = G
    for i in range(1,m):
```

CODIGO FUENTE DE LA RED SOM (SECUENCIAL)

```
45      for i in range(1,m):
46          for j in range(1,n):
47              A = np.asarray(temp[i-1:i+2,j-1:j+2]).reshape(-1)
48              salida[i-1,j-1] = np.sort(A)[d+1]
49      return salida
50
51  def orientacion(patron,w):
52      Gx,Gy = sobel(patron)
53      Gx = medfilt2(np.array(Gx,np.float32))
54      Gy = medfilt2(np.array(Gy,np.float32))
55      m,n = Gx.shape
56      mOrientaciones = np.zeros([m//w,n//w],np.float32)
57      for i in range(m//w):
58          for j in range(n//w):
59              YY = sum(sum(2*Gx[i*w:(i+1)*w,0:1]*Gy[i*w:(i+1)*w,0:1]))
60              XX = sum(sum(Gx[i*w:(i+1)*w,0:1]**2-Gy[i*w:(i+1)*w,0:1]**2))
61              mOrientaciones[i,j] = (0.5*math.atan2(YY,XX) + math.pi/2.0)*(18.0/math.pi)
62      return mOrientaciones
63
64  def kohonen_salidas(E,W):# E: representativo,W: pesos
65      M = np.zeros([14,14],np.float32)
66      m = 14
```

CODIGO FUENTE DE LA RED SOM (SECUENCIAL)

```
66      m = 14
67      for x in range(2,m-2):
68          for y in range(2,m-2):
69              w = W[:,x,y]
70              num = sum(E*w)
71              denom = math.sqrt(sum(E**2)*sum(w**2))
72              if num==0 or denom==0:
73                  M[x,y] = 0.0
74              else:
75                  M[x,y] = num/denom
76      return M
77  def kohonen_reforzamiento(M1,M_t,alpha_max,E,W):
78      (yy,xx) = np.unravel_index(np.argmax(M1,axis=None),M1.shape)
79      #print(yy,xx)
80      #print(M1[yy,xx])
81      t = M_t[xx+2,yy+2] + 1
82      M_t[xx+2,yy+2] = t
83      alpha = alpha_max/t
84      i = 0
85      for x in range(xx-1,xx+3):
86          j = 0
87          for y in range(yy-1,yy-3):
```

CODIGO FUENTE DE LA RED SOM (SECUENCIAL)

```
87              for y in range(yy-1,yy-3):
88                  w = W[:,x,y]
89                  vg = vecindad_gaussiana[i,j]
90                  d = E.T - w
91                  W[:,x,y] = w + alpha*d*vg# actualizando los pesos sinapticos
92                  j = j + 1
93              i = i + 1
94          return M_t,W,xx-2,yy-2
95
96
97  def representativo(archivo):
98      im = Image.open(archivo)
99      m,n = im.size# 256x256
100     imarray = np.array(im,np.float32)# convertir de imagen a numpy
101     patron = imarray[1:m-1,1:n-1]# 254x254
102     EE = orientacion(patron,14)# 18x18
103     return np.asarray(EE).reshape(-1)# 1x324
104
105 def mapear(archivo,W,M_index):
106     E = representativo(archivo)
107     M = kohonen_salidas(E,W)
108     (yy,xx) = np.unravel_index(np.argmax(M,axis=None),M.shape)
```

CODIGO FUENTE DE LA RED SOM (SECUENCIAL)

```
108         (yy,xx) = np.unravel_index(np.argmax(M,axis=None),M.shape)
109         x = xx - 2
110         y = yy - 2
111         tipo = M_index[x,y]
112         print(x,y,tipo)
113         if tipo==LEFT_LOOP:
114             print("LEFT_LOOP")
115         if tipo==RIGHT_LOOP:
116             print("RIGHT_LOOP")
117         if tipo==WHORL:
118             print("WHORL")
119         if tipo==ARCO:
120             print("ARCO")
121
122
123 print("----------------------")
124 print("creando los patrones de entrada")
125 EEE = np.zeros([15,324],np.float32)
126 EEE[0,:] = representativo('arco1.tif')
127 EEE[1,:] = representativo('whorl1.tif')
128 EEE[2,:] = representativo('right1.tif')
129 EEE[3,:] = representativo('left1.tif')
```

CODIGO FUENTE DE LA RED SOM (SECUENCIAL)

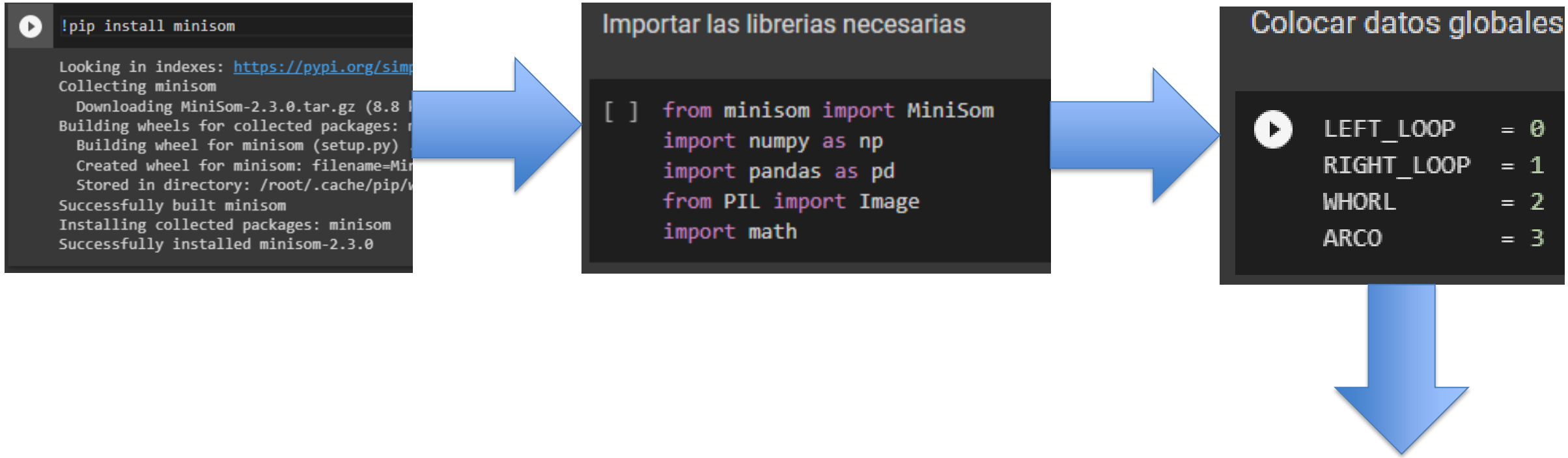
```
129 EEE[3,:] = representativo('left1.tif')
130 EEE[4,:] = representativo('arco2.tif')
131 EEE[5,:] = representativo('whorl2.tif')
132 EEE[6,:] = representativo('right2.tif')
133 EEE[7,:] = representativo('left2.tif')
134 EEE[8,:] = representativo('arco3.tif')
135 EEE[9,:] = representativo('whorl3.tif')
136 EEE[10,:] = representativo('right3.tif')
137 EEE[11,:] = representativo('left3.tif')
138 EEE[12,:] = representativo('arco4.tif')
139 EEE[13,:] = representativo('whorl4.tif')
140 EEE[14,:] = representativo('right4.tif')
141 # Inicia el proceso de entrenamiento de la RN
142 print("----------------------")
143 print("iniciando el aprendizaje")
144 for iter in range(max_iter):
145     print(iter)
146     for i in range(15):
147         E = EEE[i,:]
148         M = kohonen_salidas(E,W)
149         M_t,W,x,y = kohonen_reforzamiento(M,M_t,alpha_max,EEE[0,:],W)
150         if i in [0,4,8,12]:
```

CODIGO FUENTE DE LA RED SOM (SECUENCIAL)

```
149             M_t,W,x,y = kohonen_reforzamiento(M,M_t,alpha_max,EEE[0,:],W)
150             if i in [0,4,8,12]:
151                 M_index[x,y] = ARCO
152             elif i in [3,7,11]:
153                 M_index[x,y] = LEFT_LOOP
154             elif i in [2,6,10,14]:
155                 M_index[x,y] = RIGHT_LOOP
156             elif i in [1,5,9,13]:
157                 M_index[x,y] = WHORL
158 print("Fin del aprendizaje")
159 print("----------------------")
160 print("Matriz de indices o representacion de lo aprendido: ")
161 print(M_index)
162 print("----------------------")
163 print("Realizar el mapeo o comprobación: ")
164 mapear('left1.tif',W,M_index)
165 #print(EEE)
166 #print(EEE.shape)
```

EJEMPLO EN ENTORNO COLAB:

- Clasificación de huella

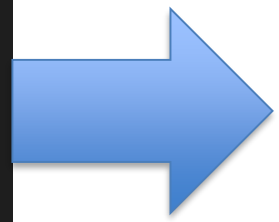


```
[ ]
    def sobel(I):
        m,n = I.shape# I de 254x254
        Gx = np.zeros([m-2,n-2],np.float32)# Gx de 252x252
        Gy = np.zeros([m-2,n-2],np.float32)# Gy de 252x252
        gx = [[-1,0,1],[ -2,0,2],[ -1,0,1]]
        gy = [[1,2,1],[ 0,0,0],[ -1,-2,-1]]
        for j in range(1,m-2):
            for i in range(1,n-2):
                Gx[j-1,i-1] = sum(sum(I[j-1:j+2,i-1:i+2]*gx))
                Gy[j-1,i-1] = sum(sum(I[j-1:j+2,i-1:i+2]*gy))
        return Gx,Gy

    def medfilt2(G,d=3):
        m,n = G.shape
        temp = np.zeros([m+2*(d//2),n+2*(d//2)],np.float32)
        salida = np.zeros([m,n],np.float32)
        temp[1:m+1,1:n+1] = G
        for i in range(1,m):
            for j in range(1,n):
                A = np.asarray(temp[i-1:i+2,j-1:j+2]).reshape(-1)
                salida[i-1,j-1] = np.sort(A)[d+1]
        return salida

    def orientacion(patron,w):
        Gx,Gy = sobel(patron)# patron de 254x254
        Gx = medfilt2(Gx)# Gx de 252x252
        Gy = medfilt2(Gy)# Gy de 252x252
        m,n = Gx.shape
        mOrientaciones = np.zeros([m//w,n//w],np.float32)# de una matriz de 18x18
        for i in range(m//w):
            for j in range(n//w):
                YY = sum(sum(2*Gx[i*w:(i+1)*w,j:j+1]*Gy[i*w:(i+1)*w,j:j+1]))
                XX = sum(sum(Gx[i*w:(i+1)*w,j:j+1]**2-Gy[i*w:(i+1)*w,j:j+1]**2))
                #YY = sum(sum(2*Gx[i*w:(i+1)*w,0:1]*Gy[i*w:(i+1)*w,0:1]))
                #XX = sum(sum(Gx[i*w:(i+1)*w,0:1]**2-Gy[i*w:(i+1)*w,0:1]**2))
                mOrientaciones[i,j] = (0.5*math.atan2(YY,XX) + math.pi/2.0)*(180.0/math.pi)
        return mOrientaciones
    def representativo(archivo):
        im = Image.open(archivo)
        m,n = im.size
        imarray = np.array(im,np.float32)
        patron = imarray[1:m-1,1:n-1]# de 256x256  a 254x254
        EE = orientacion(patron,14)# retorna EE de 18x18
        return np.asarray(EE).reshape(-1)
```

Vamos a importar los datos

```
[ ]  Xi = np.zeros([15,324],np.float32)# dimension de 15x324
     Xi[0,:] = representativo('arco1.tif')
     Xi[1,:] = representativo('whorl1.TIF')
     Xi[2,:] = representativo('Right1.tif')
     Xi[3,:] = representativo('left1.tif')
     Xi[4,:] = representativo('Arco2.tif')
     Xi[5,:] = representativo('whorl2.tif')
     Xi[6,:] = representativo('Right2.tif')
     Xi[7,:] = representativo('left2.tif')
     Xi[8,:] = representativo('Arco3.tif')
     Xi[9,:] = representativo('whorl3.tif')
     Xi[10,:] = representativo('Right3.tif')
     Xi[11,:] = representativo('left3.tif')
     Xi[12,:] = representativo('Arco4.tif')
     Xi[13,:] = representativo('whorl4.tif')
     Xi[14,:] = representativo('Right4.tif')
     print(Xi.shape)
     #Yi = [4,3,2,1,4,3,2,1,4,3,2,1,4,3,2]-1
     Yi = [3,2,1,0,3,2,1,0,3,2,1,0,3,2,1]
     print(len(Yi))
```

```
(15, 324)
15
```

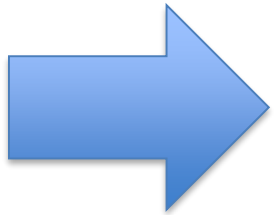


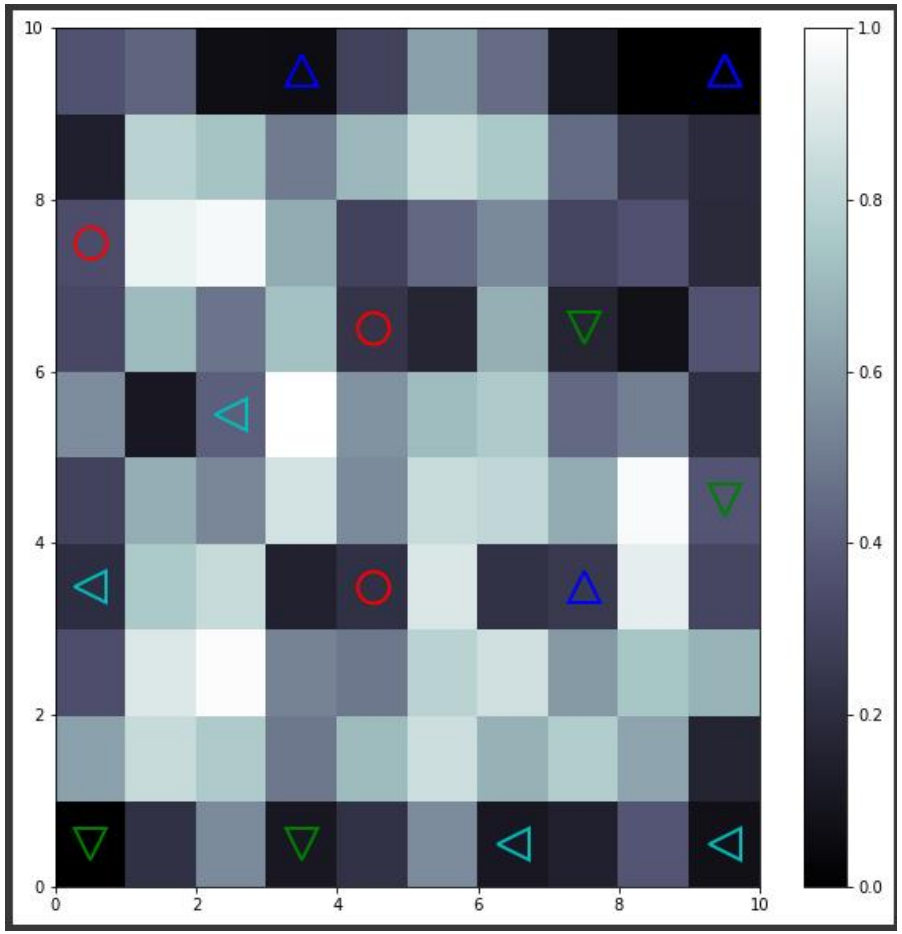
Usar la libreria SOM

```
[ ]  max_iter = 50000
     alpha_max = 0.5

     som = MiniSom(x=10, y=10, input_len=324, sigma=1.0, learning_rate=alpha_max,activation_distance='euclidean',
                   topology='hexagonal', neighborhood_function='gaussian')
     som.random_weights_init(Xi)
     som.train_random(data=Xi, num_iteration=max_iter)
```

Visualizar los datos en la matriz index

```
[ ]  from pylab import bone, pcolor, colorbar, plot, show
     from pylab import rcParams
     rcParams['figure.figsize'] = 10, 10
     bone()
     pcolor(som.distance_map().T)
     colorbar()
     valores = [0,1,2,3]
     M = np.zeros([10,10],np.float32)-1
     markers = ['o', 'v', '^', '<']
     colors =  ['r', 'g', 'b', 'c']
     for i, x in enumerate(Xi):
         w = som.winner(x)
         plot(w[0] + 0.5,
              w[1] + 0.5,
              markers[Yi[i]],
              markeredgecolor=colors[Yi[i]],
              markerfacecolor='None',
              markersize=20,
              markeredgewidth=2)
         M[w[0],w[1]] = Yi[i]
     show()
```

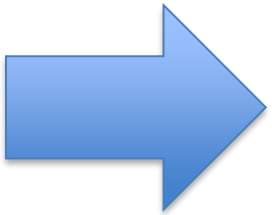




Comprobacion

```
#XX = X[1,:]
B = representativo('arco1.tif')
#XX = Xi[7,:] # representativo('arco1.tif') # arco
w = som.winner(B)
print(w)
print(N[w])#
```

```
(9, 0)
AR
```



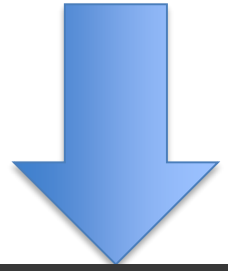
```
[ ]  Y_simulado = []
     for i in range(len(Yi)):
       XX = Xi[i,:]
       w = som.winner(XX)
       if N[w]=="AR":
         Y_simulado.append(3)
       elif N[w]=="WH":
         Y_simulado.append(2)
       elif N[w]=="RL":
         Y_simulado.append(1)
       elif N[w]=="LL":
         Y_simulado.append(0)
       else:
         Y_simulado.append(-1)
     print(Y_simulado)
```

```
[3, 2, 1, 0, 3, 2, 1, 0, 3, 2, 1, 0, 3, 2, 1]
```

Evaluar precision

```
precision = 0
for i in range(len(Yi)):
  if Yi[i]==Y_simulado[i]:
    precision = precision + 1
precision = precision/float(len(Yi))
print(precision)
```

```
1.0
```



```
[ ] from sklearn.metrics import confusion_matrix
    Matriz_de_Confusion = confusion_matrix(Yi, Y_simulado)
    Matriz_de_Confusion
    """sensibilidad"""
    sensibilidad = (Matriz_de_Confusion[0,0])/np.sum(Matriz_de_Confusion[0,0]+ Matriz_de_Confusion[1,0])
    sensibilidad
```

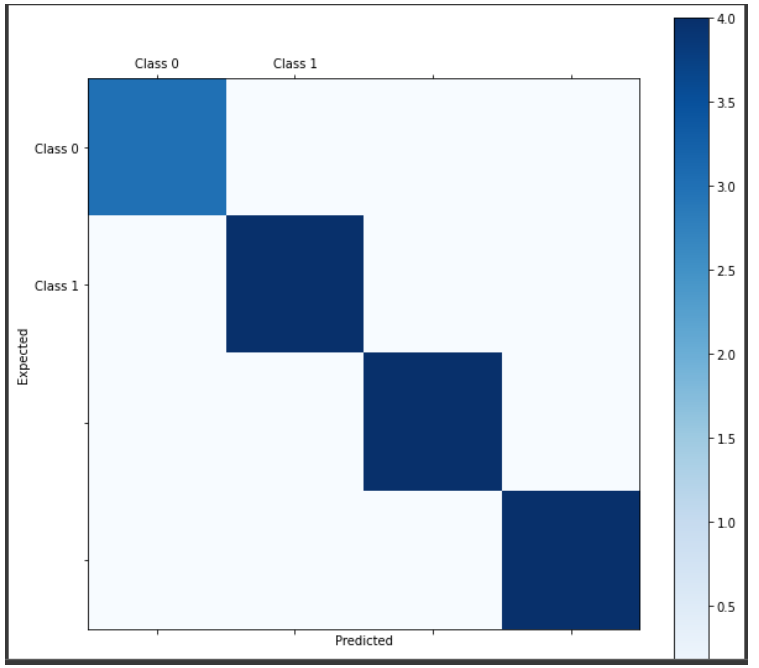
```
1.0
```

```
[ ] """especificidad"""
    especificidad = (Matriz_de_Confusion[1,1])/np.sum(Matriz_de_Confusion[1,1]+ Matriz_de_Confusion[0,1])
    especificidad
```

```
1.0
```

```python
from sklearn.metrics import confusion_matrix
from matplotlib import pyplot as plt
from sklearn.metrics import classification_report

conf_mat = confusion_matrix(y_true=Yi, y_pred=Y_simulado)
print('Matriz de Confusión - DATOS ORIGINALES:\n', conf_mat)
print('Métricas de Matriz de Confusión - DATOS ORIGINALES:\n',classification_report(Yi,Y_simulado))
labels = ['Class 0', 'Class 1']
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(conf_mat, cmap=plt.cm.Blues)
fig.colorbar(cax)
ax.set_xticklabels([''] + labels)
ax.set_yticklabels([''] + labels)
plt.xlabel('Predicted')
plt.ylabel('Expected')
plt.show()
```

```
Matriz de Confusión - DATOS ORIGINALES:
 [[3 0 0 0]
 [0 4 0 0]
 [0 0 4 0]
 [0 0 0 4]]
Métricas de Matriz de Confusión - DATOS ORIGINALES:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         3
           1       1.00      1.00      1.00         4
           2       1.00      1.00      1.00         4
           3       1.00      1.00      1.00         4

    accuracy                           1.00        15
   macro avg       1.00      1.00      1.00        15
weighted avg       1.00      1.00      1.00        15
```



Guardar lo aprendido

```
[ ]  import pickle
     # almacenando la red som en un archivo som.p
     with open('som.p', 'wb') as outfile:
         pickle.dump(som, outfile)
```

Leer la red som desde un archivo

```
[ ]  with open('som.p', 'rb') as infile:
         som = pickle.load(infile)
```

```
# Guardar lo aprendido
import pickle
with open('som.pkl','wb') as outfile:
  pickle.dump(som,outfile)
```

CONSIDERACIONES PARA EJECUCION EN PARALELO

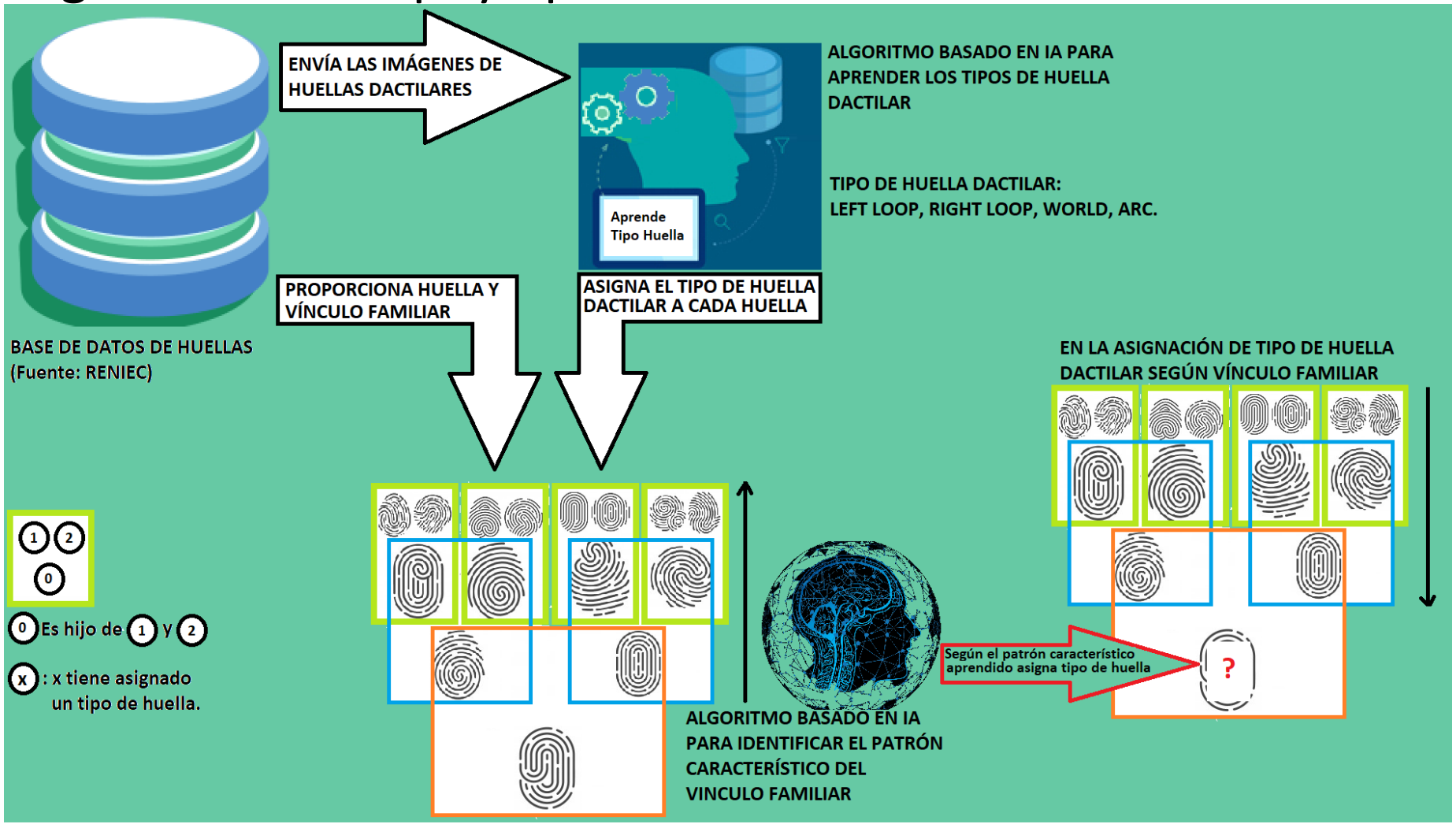
```python
from concurrent.futures import ProcessPoolExecutor
import time

def square(number):
    time.sleep(0.1) # Simulate some work
    return number * number

if __name__ == '__main__':
    numbers = range(10)
    with ProcessPoolExecutor() as executor:
        results = executor.map(square, numbers)
        for result in results:
            print(result)
```

PARALELIZAR

```
def kohonen_salidas(E,W):# E: representativo,W: pesos
    M = np.zeros([14,14],np.float32)
    m = 14
    for x in range(2,m-2):
        for y in range(2,m-2):
            w = W[:,x,y]
            num = sum(E*w)
            denom = math.sqrt(sum(E**2)*sum(w**2))
            if num==0 or denom==0:
                M[x,y] = 0.0
            else:
                M[x,y] = num/denom
    return M
```

RECOMENDACIONES:

- Dermatoglypics, Patrón de huellas dactilares del vínculo familiar consanguíneo como apoyo para identificar el ADN



TAREA :

TAREA

1) Crear los patrones de entrada y su representativo en orientaciones. en dimensión de 1x324 por cada huella

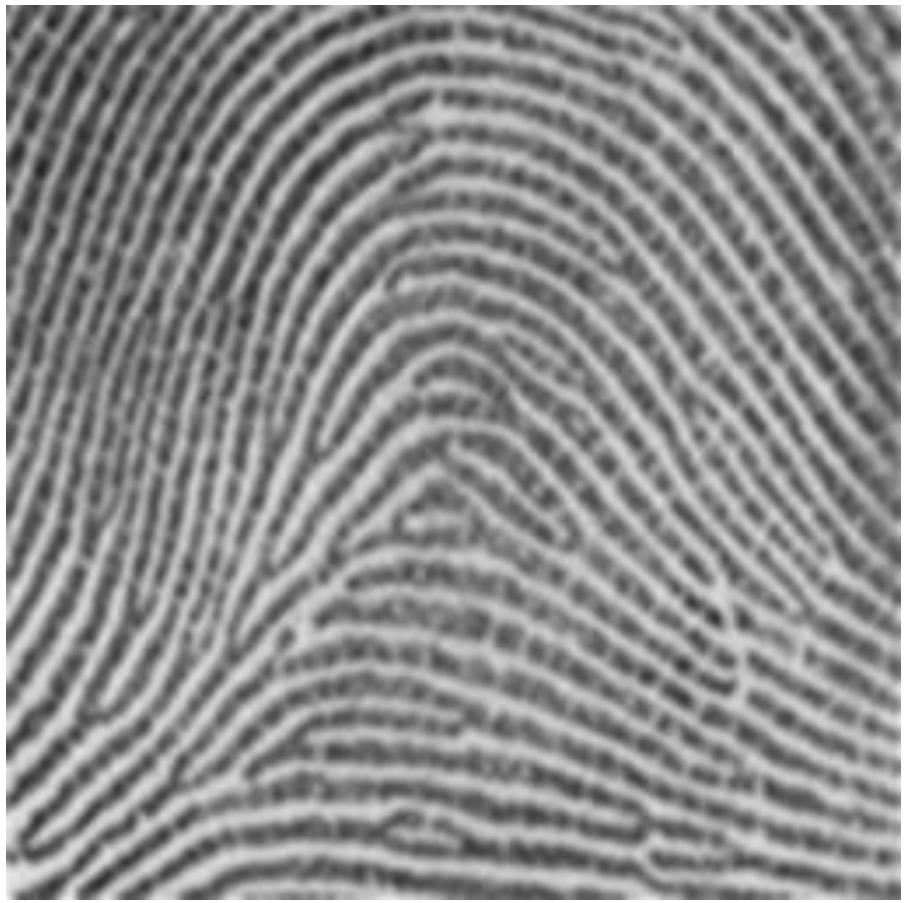
2) Crear los pesos sinápticos por cada entrada asociada. considerar la dimensión de la capa competitiva de 10x10.

3) Realizar el proceso de aprendizaje usando procesamiento paralelo.

```
def kohonen_salidas(E,W):# E: representativo,W: pesos
    M = np.zeros([14,14],np.float32)
    m = 14
    for x in range(2,m-2):
        for y in range(2,m-2):
            w = W[:,x,y]
            num = sum(E*w)
            denom = math.sqrt(sum(E**2)*sum(w**2))
            if num==0 or denom==0:
                M[x,y] = 0.0
            else:
                M[x,y] = num/denom
    return M
```

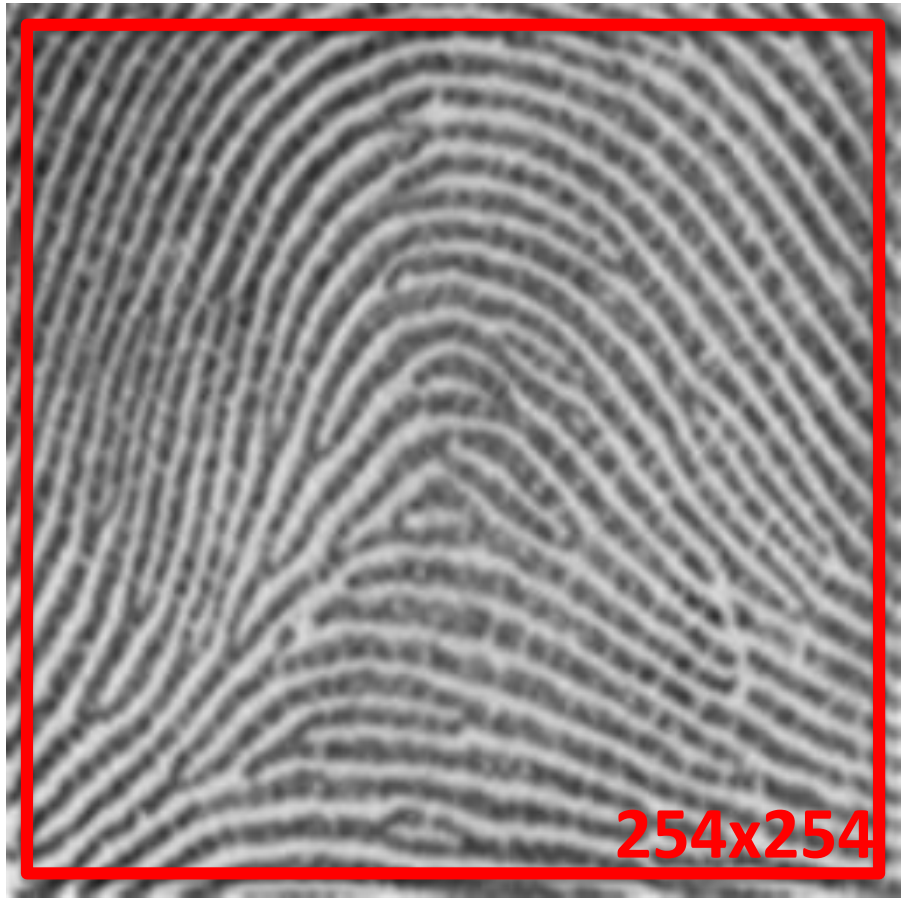
TAREA GRUPAL:

Ejemplo la huella
Arco1.tif (de dimensión 256x256)



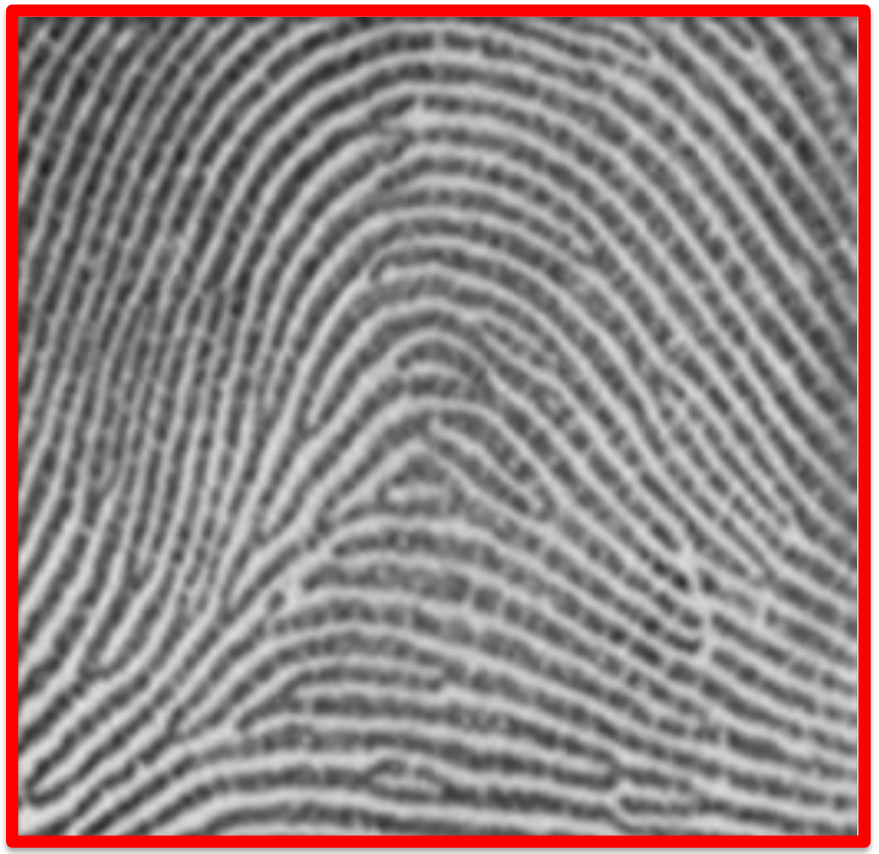
TAREA GRUPAL:

Ejemplo la huella Arco1.tif que tiene la dimensión de 256x256



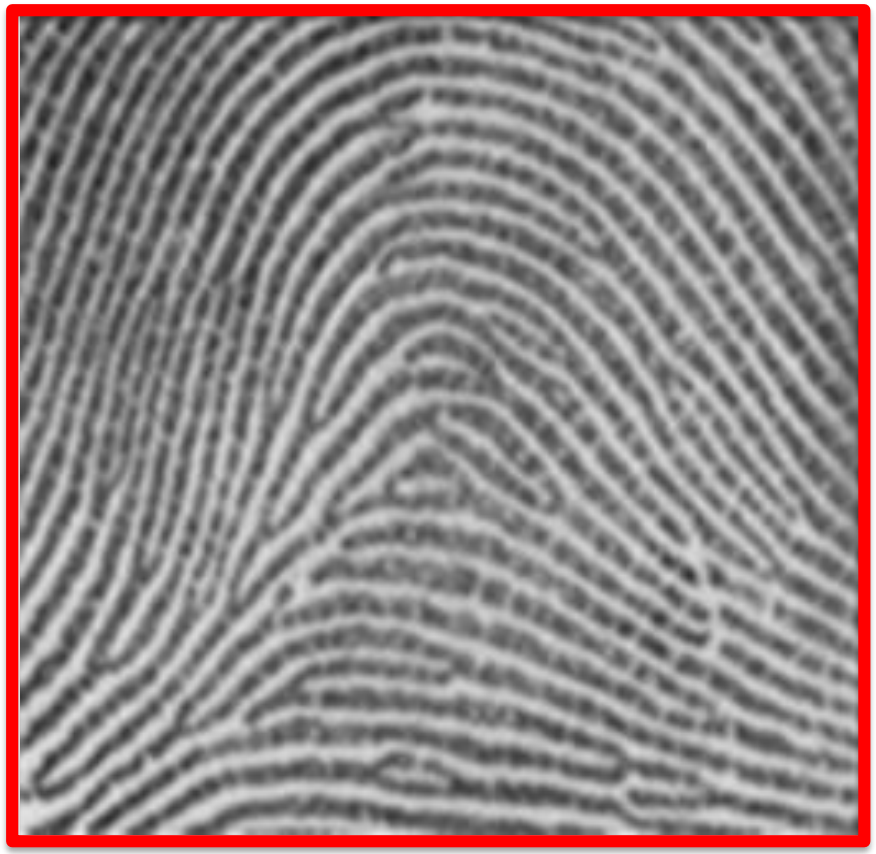
256x256

Se recorta la imagen, tal que se pierde 2 columnas, así también 2 filas.



patron 254x254

TAREA GRUPAL:



patron 254x254

Se le aplica la función de orientación a esta matriz en proporciones de 14x14 para obtener una matriz de orientación de 18x18

Aquí la función sobel en python

Donde:
Imagen es donde se va a obtener la función sobel (ósea para este caso es el patron de dimensión 254x254).

```
def sobel(patron):
    m,n = patron.shape
    I = np.array(patron,np.float16)
    Gx = np.zeros([m-2,n-2])
    Gy = np.zeros([m-2,n-2])
    gx = [[-1,0,1],[ -2,0,2],[ -1,0,1]]
    gy = [[1,2,1],[ 0,0,0],[ -1,-2,-1]]
    for i in range(1,m-1):
        for j in range(1,n-1):
            Gx[i-1,j-1] = sum(sum(I[i-1:i+2,j-1:j+2]*gx))
            Gy[i-1,j-1] = sum(sum(I[i-1:i+2,j-1:j+2]*gy))

    return Gx,Gy
```

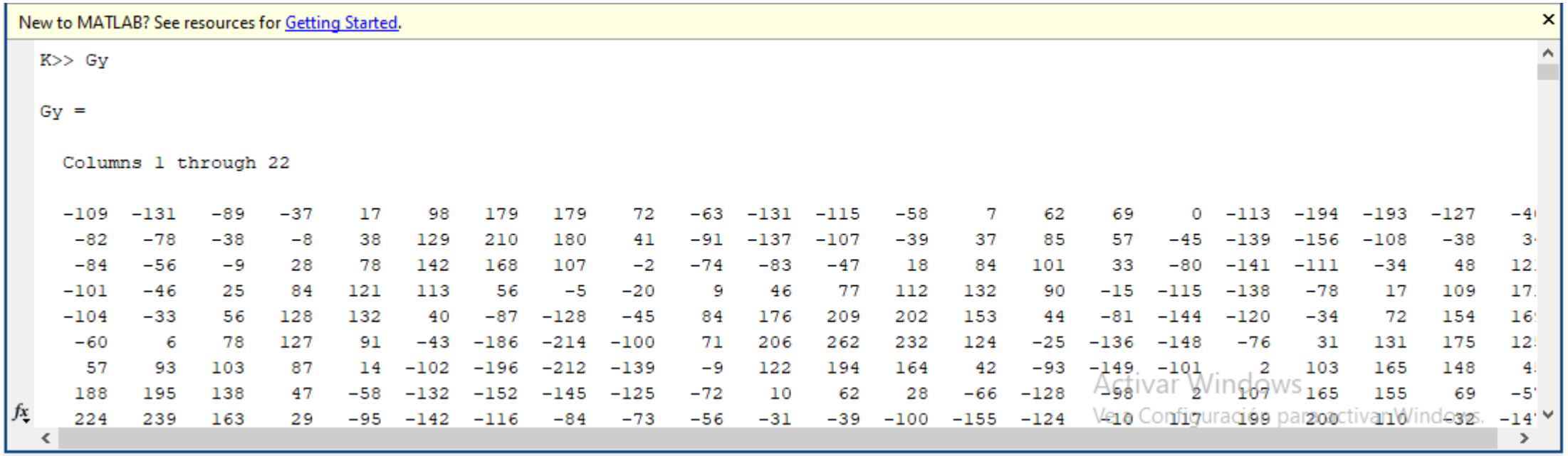
TAREA GRUPAL:

Gx en la línea 9

```
Gx =

  Columns 1 through 22

   167   225    91   -43   -75   -66  -103  -135   -16   161   197   109     6   -97  -200  -239  -112   129   274   225    75   -6
   220   190    44   -78  -140  -173  -158   -50   145   267   205    55   -65  -167  -233  -185     5   219   282   168    -4  -14
   238   140     7  -108  -206  -262  -176    59   284   332   195    -1  -142  -232  -233   -89   124   255   233    92   -80  -21
   221    74   -45  -152  -249  -275  -132   137   330   329   170   -43  -200  -260  -180    31   217   246   152     1  -163  -26
   176    -7  -124  -204  -238  -186   -37   144   255   246   124   -61  -210  -221   -68   149   262   200    52  -100  -236  -27
   116   -84  -208  -241  -169   -29    76   100   102    99    44   -70  -164  -110    79   238   248   118   -59  -195  -267  -20
    55  -125  -249  -233   -68   120   164    56   -41   -59   -64   -90   -84    40   213   271   179    12  -161  -255  -234   -8
     0  -113  -214  -161    38   204   194    35  -113  -170  -166  -112     2   170   280   230    72   -89  -219  -249  -137    4
   -58   -77  -117   -33   139   224   168    22  -131  -222  -221  -101    86   235   246   120   -45  -159  -208  -160     2   14
  -128   -60    -2   115   224   211   111   -12  -152  -239  -201   -31   161   225   128   -24  -142  -187  -136    -8   145   19
  -198   -69    97   239   270   176    36   -81  -192  -223  -108    79   200   153   -15  -149  -199  -173   -34   148   251   19
  -243   -71   173   308   259   113   -50  -165  -217  -157    20   174   187    57  -122  -224  -218  -126    64   247   287   17
  -236   -26   228   308   191    26  -129  -222  -190   -44   133   209   137   -15  -169  -254  -210   -56   138   258   239   12
  -162    66   248   237    81   -66  -177  -218  -106    84   201   181    71   -57  -179  -249  -169    32   183   190   130    7
```

TAREA GRUPAL:

Gy en la línea 9



TAREA GRUPAL:

Gx en la línea 10

```
K>> Gx

Gx =

  Columns 1 through 22

    0    91     0     0   -66   -75   -66   -16     0   145   109     6     0   -65   -167   -112     0     5   168    75     0    -
  167   167    44   -75   -108  -158  -135   -50   145   197   195    55   -65   -167  -200   -185     5   219   225   168    -4   -14
  140   140     7  -108   -173  -176  -158    59   267   267   195    -1  -142   -200  -185    -89   124   219   219    92   -80   -21
   74    74   -45  -152   -206  -206  -132   137   255   255   170   -43  -200   -210  -180     31   200   217   152     1  -163   -23
    0    -7  -124  -204   -204  -169   -29   102   144   170    99   -61  -164   -180   -68    149   217   200    52  -100  -207   -23
    0   -84  -204  -208   -186   -37    76   100   100    99   -59   -70   -90    -84    79    213   200   118   -59  -195  -234   -20
    0  -113  -208  -208    -68    76   100    76    35   -59   -70   -84   -84     40   213    230   179    12  -161  -219  -207    -8
    0  -113  -125  -117     38   164   164    35   -59  -131  -112   -90     2    170   230    213    72   -89  -161  -208  -137     4
  -58   -77   -77    -2    139   194   168    22  -131  -170  -170  -101    86    170   225    120   -45  -142  -160  -137     2    14
```

TAREA GRUPAL:

Gy en la línea 10

```
K>> Gy

Gy =

  Columns 1 through 22

    0   -78   -37     0     0    38   129    72     0     0   -91   -58     0     0    37     0     0   -45   -113   -108   -38
  -78   -82   -38    -8    38   129   168   168    41   -74   -91   -83   -39    37    62    57   -45   -113   -139   -111   -38
  -56   -56    -9    28    84   121   129    56    -2   -20   -74   -39    37    85    84    33   -80   -115   -111    -38    34
  -46   -46    25    78   113   113    56    -5    -5    -2    46    77   112   101    84   -15  -115   -115    -78     17   109
  -33   -33    56    91   113    56   -43   -87   -20    46    84   202   153   124    44   -81  -120   -115    -34     72   131
    0    56    87    91    87   -43  -128  -139  -100    71   176   202   194   124   -25  -101  -120    -76     31    131   148
    6    93    93    87    14  -102  -152  -152  -125    -9    71   164   124    28   -93  -101   -98      2    107    148   131
   93   163   103    47   -58  -116  -142  -139   -84   -56    -9    28    28   -93   -98   -98     2    107    155    148    69
  174   174   138    29   -95  -120  -132  -101   -72   -31   -31   -31   -87  -124   -98   -10   107    176    165    110   -32
```