



---

# MEMORIA IA II PRÁCTICA 1

---

Grupo 5: Julián Prieto Velasco, Javier Muñoz Rojas, Guillermo Ramírez  
Cárdenas, Pedro José Paniagua Falo



20 DE FEBRERO DE 2023

INTELIGENCIA ARTIFICIAL II  
Universidad Francisco de Vitoria





1. Introducción .....	6
2. Objetivos .....	6
3. Práctica 1 .....	7
3.1 Introducción .....	7
3.2 Implementación .....	7
3.2.1 Asignación de variables .....	7
3.2.2 Funciones para el entrenamiento .....	9
3.2.3 Entrenamiento .....	13
3.2.4 Clasificación .....	14
3.2.5 Prueba .....	15
3.3 Cuestiones .....	17
3.3.1 Cuestión 1 .....	17
3.3.2 Cuestión 2 .....	20
3.3.3 Cuestión 3 .....	22
3.4- Conclusiones .....	25
4.- Práctica 2 .....	26
4.1- Introducción .....	26
4.2- Implementación .....	26
4.2.1 Entrenamiento .....	27
4.2.2- Clasificación .....	27
4.3- Cuestiones .....	28
4.3.1 Cuestión 1 .....	28
4.4.2 Cuestión 2 .....	32
4.3.3 Cuestión 3 .....	33
4.4.4 Cuestión 4 Optativa .....	35
4.5 Conclusiones .....	38
5.- Anexo A .....	39
6.- Bibliografía .....	49

## Tabla de Ilustraciones

Ilustración 1 Red de Kohonen .....	6
Ilustración 2 Asignación de las variables de inicio .....	7
Ilustración 3 Creación de las variables definidas por teclado .....	7
Ilustración 4 Tabla RGB .....	8
Ilustración 5 Matriz tridimensional .....	8
Ilustración 6 Datos de entrada del Mapa .....	8
Ilustración 7 Función de calcular BMU .....	9



Ilustración 8 Función para la variación del learning rate .....	9
Ilustración 9 Función de variación de eta .....	10
Ilustración 10: Variación lineal de la eta .....	10
Ilustración 11 Función de amortiguación.....	10
Ilustración 12: Función Calcular MSE .....	11
Ilustración 13 Función pintarmapa() .....	11
Ilustración 14 Ejemplo de salida de pintarmapa() .....	11
Ilustración 15 Función de animación .....	12
Ilustración 16 Instalación de imageio.....	12
Ilustración 17 Función para pintar la variación del learning rate .....	12
Ilustración 18 Código de aprendizaje de la IA.....	14
Ilustración 19 Código de clasificación .....	14
Ilustración 20: código mapa clasificación.....	15
Ilustración 21 código mapa activación.....	15
Ilustración 22 código mapa distancias .....	15
Ilustración 23: Código Som de Prueba .....	16
Ilustración 24: RGB de pesos tras 10000 interacciones con 0.0001 eta y 30 de lado.....	17
Ilustración 25: RGB de pesos tras 3000 interacciones con 0.005 eta y 30 de lado.....	17
Ilustración 26 Entrenamiento con 30 de lado, 10000 iteraciones y 0.0001 eta .....	18
Ilustración 27: Variación del aprendizaje a lo largo del entrenamiento.....	18
Ilustración 28 Curva de aprendizaje con 10000 iteraciones, 30 de lado y eta 0,001.....	19
Ilustración 29 Curva de aprendizaje con 10000 iteraciones, 30 de lado y eta 0,01.....	19
Ilustración 30 Curva de aprendizaje con 10000 iteraciones, 30 de lado y eta 0,0001.....	19
Ilustración 31 Matriz de pesos para t(0) .....	20
Ilustración 32 Matriz de pesos para t(10000) .....	20
Ilustración 33 Mapa de clasificación tras el entrenamiento .....	21
Ilustración 34 Mapa de activaciones al final del entrenamiento .....	21
Ilustración 35: Código colores de entrenamiento declaración .....	22
Ilustración 36 Código colores de cuestión 3 entrenamiento .....	23
Ilustración 37 Mapas de clasificación, activaciones, distancias y número de clases, error de cuantificación y error topológico de las pruebas .....	23
Ilustración 38 Mapa de clasificación de la prueba .....	24
Ilustración 39 Histograma 3D de activación de la prueba .....	24
Ilustración 40 Importación de librerías .....	26
Ilustración 41 Lectura del CSV y guardado de datos.....	26
Ilustración 42 Entrenamiento Práctica 2.....	27
Ilustración 43 Clasificación Práctica 2 .....	28
Ilustración 44 Curva de aprendizaje con Lado de mapa 20, iteraciones 5000 y eta 0.0001.....	29
Ilustración 45 Curva de aprendizaje con Lado de mapa 20, 5000 iteraciones y 0.001 eta.....	29
Ilustración 46 Curva de aprendizaje con Lado de mapa 15, 2500 iteraciones y 0.01 de eta.....	30
Ilustración 47 Curva de aprendizaje con Lado de mapa 15, 2500 iteraciones y 0.001 eta.....	30
Ilustración 48 Variación del Learning Rate con Lado de mapa 20, iteraciones 5000 y eta 0.0001 .....	31
Ilustración 49 Curva de aprendizaje con Lado de mapa 20, 2500 iteraciones y 0.1 eta.....	31
Ilustración 50 Histograma del mapa de Activaciones de la Práctica 2.....	32
Ilustración 51 Mapa de Clasificación según GPD .....	34
Ilustración 52 Mapa de clasificación según Mortality rate, neonatal.....	35
Ilustración 53 Guardar datos del mapa.....	36



Ilustración 54 Creación y rellenado del diccionario de datos de world .....	36
Ilustración 55 Creación y coloreo del mapamundi .....	37
Ilustración 56 Mapamundi con sus neuronas .....	37

## Tabla de Ecuaciones

Ecuación 1 Distancia Euclídea .....	9
Ecuación 2 Variación de eta .....	10
Ecuación 3 Variación de vecindario.....	10
Ecuación 4 Amortiguación.....	10
Ecuación 5 Distancia Euclídea .....	15



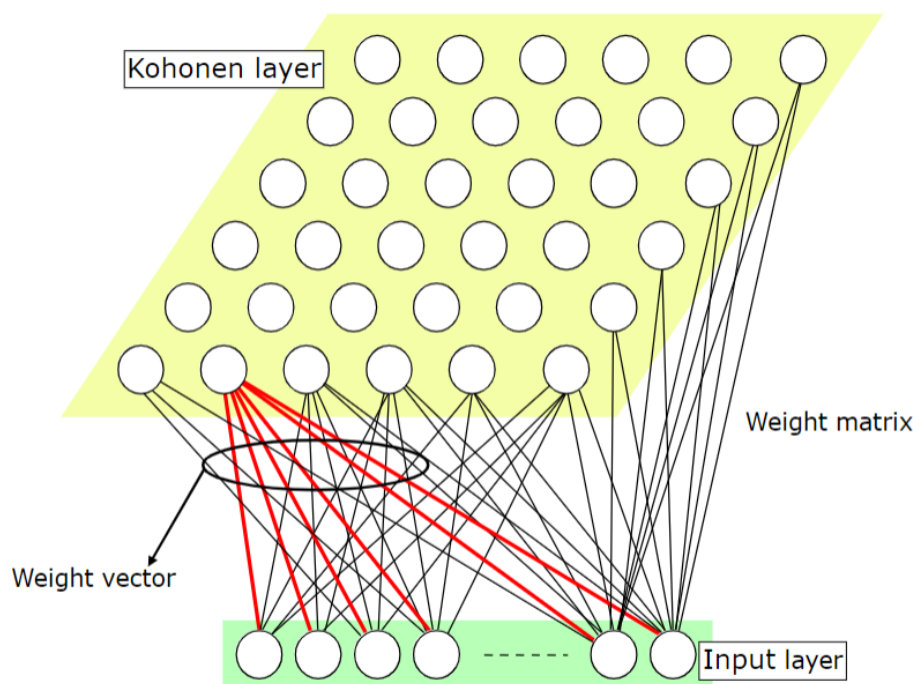


## 1. Introducción

Las redes de mapas autoorganizados (SOM) son un método de aprendizaje no supervisado para explorar y visualizar datos complejos. Este tipo de red neuronal artificial está diseñada para encontrar relaciones importantes entre los datos y agruparlos en categorías o grupos. En este experimento, la red SOM se utilizará para clasificar dos conjuntos de datos en diferentes categorías.

Para ello, se entrena la red con un conjunto de datos de entrada y se ajustan los parámetros de la red para que las neuronas de la capa de salida puedan agrupar los datos de manera eficiente. Luego, se analizará la distribución de los datos en el espacio de características creado por la red SOM para comprender mejor las relaciones entre los datos y su agrupación.

Visualizando la red de manera gráfica se obtendría el siguiente dibujo:



*Ilustración 1 Red de Kohonen*

Donde todas las neuronas de entrada se encuentran conectadas con las neuronas de la capa de Kohonen en una matriz de múltiples dimensiones y a través de un aprendizaje competitivo donde hay una neurona ganadora y un vecindario que aprenden a organizar los elementos.

## 2. Objetivos

El propósito de esta práctica de laboratorio es comprender cómo funciona la red SOM y cómo se puede usar para clasificar datos en diferentes categorías. También se investigarán las propiedades de las redes SOM y su impacto en la calidad de la clasificación de datos.



### 3. Práctica 1

#### 3.1 Introducción

En esta parte de la práctica se ha solicitado realizar un clasificador de colores RGB, usando una red SOM. Para ello se ha utilizado el lenguaje de programación Python, con la distribución de anaconda y las librerías numpy, \_\_future\_\_, matplotlib, math, pprint, imageio, math, pprint, os.

#### 3.2 Implementación

##### 3.2.1 Asignación de variables

Una vez entendido el funcionamiento de una red SOM con aprendizaje competitivo comenzamos definiendo las principales variables para obtener el Dataset de entrenamiento. Como usamos 3 valores RGB y estos varían entre 0 y 255 declaramos las variables con estos datos. Creamos una matriz llamada datos de 100 colores al azar usando el código RGB.

```
# Código para obtener el Dataset que se va a usar en el entrenamiento
valor_min = 0
valor_max = 255
valores_color = 3
num_colores = 100
datos = np.random.randint(valor_min, valor_max, (valores_color, num_colores))
print(datos)
```

Ilustración 2 Asignación de las variables de inicio

Definimos por teclado al iniciar el programa el tamaño del mapa, el periodo de iteraciones, el coeficiente de aprendizaje (learning rate) y si se quieren normalizar los datos o no (0 es no y 1 es sí). Cuanto más alto sea el valor del coeficiente de aprendizaje más rápido aprenderá la IA, pero el inconveniente de esto es que aprenderá peor. Para que la IA entrene correctamente hay que realiza una combinación de estos valores que otorgue eficiencia y efectividad.

Se ha decidido *parsear* a *int* el lado del mapa, el periodo.

La variable *learning\_rate* debe ser un *float* para el buen aprendizaje de la red neuronal. Esta variable nos permite establecer cómo se va a ajustar cada neurona en cada periodo.

En *normalizar\_datos*, se ha decidido asignar directamente el valor de este a uno, pues la función *pintar\_mapa* que se implementa tan solo permite usar valores entre 0 y 1, de manera que es necesario normalizarlo.

```
# Inicializa tamaño del mapa de Kohonen, número de iteraciones y learning rate
# Inicializa normalizar_datos dependiendo de si tienes que normalizar los datos o no
lado_mapa = int(input("Introduzca el lado"))
periodo = int(input("Introduzca el Periodo"))
learning_rate = float(input("Introduzca el Learning Rate"))
✓ 6.4s
```

Ilustración 3 Creación de las variables definidas por teclado



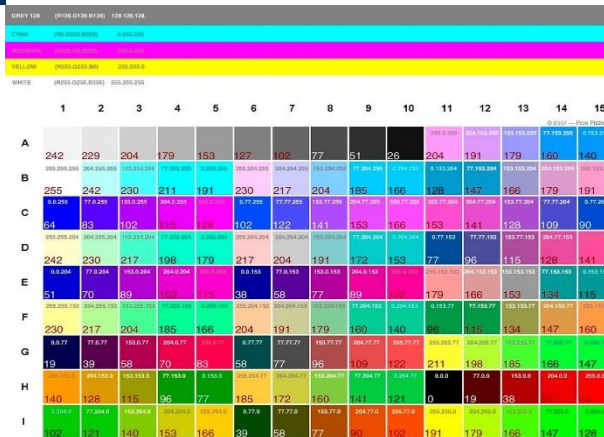


Ilustración 4 Tabla RGB

Se ha asignado el valor a *num\_entradas* de tres porque los datos que se van a clasificar son RGB (Red, Green, Blue) como se observa en la ilustración 3. Además, los valores de RGB van de 0 a 255, por lo que se asigna el número de datos a 256.

En lo que atañe al vecindario, grupo de neuronas que están conectadas topológicamente entre sí, se le ha asignado la mitad del lado del mapa.

Tras asignar el lado del mapa, se normalizan los datos para poder ser usado en la función *pintarmapa()* como ya se ha comentado anteriormente.

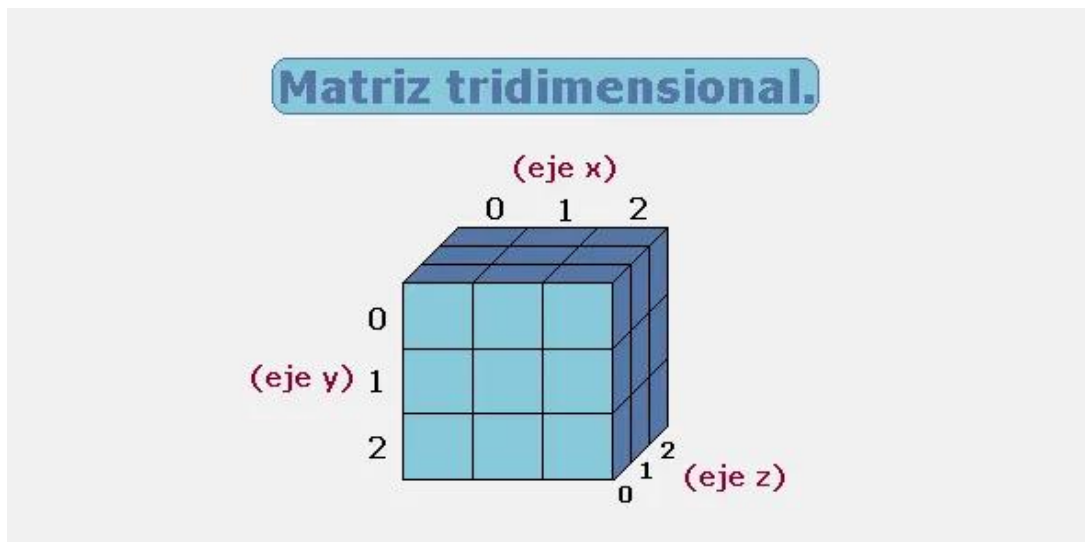


Ilustración 5 Matriz tridimensional

En el marco de la matriz de pesos, se ha creado una matriz aleatoria tridimensional, como la que observamos en la ilustración 4. Sus dimensiones serán, *lado\_mapa*, en el eje x e y, y *num\_entradas* en el eje z.

```
# Establece el numero de entradas del mapa y el número de datos que se van a usar para entrenar.
# Utiliza una función que obtenga automáticamente los valores a partir del Dataset.
num_entradas = 3 #3 por el RGB
num_datos = 256 #de 0 a 255

# Calcula el vecindario inicial. Debe ser la mitad del lado del mapa de Kohonen
vecindario = lado_mapa / 2
# Normaliza los datos si fuese necesario dividiendo cada dato por el máximo en la matriz
if normalizar_datos:
    datos = np.divide((datos), (datos.max()))
# Crea una matriz de pesos con valores random entre 0 y 1. Usa la función random.random de la librería NumPy
matriz_pesos = np.random.random(size = (lado_mapa, lado_mapa, num_entradas))
```

Ilustración 6 Datos de entrada del Mapa

En este punto se empiezan a definir las funciones para el entrenamiento y la clasificación.



### 3.2.2 Funciones para el entrenamiento

```
def calcular_bmu(patron_entrada, m_pesos, m):  
    distancias = np.linalg.norm(m_pesos - patron_entrada, axis=2)  
    bmu_idx = np.unravel_index(np.argmin(distancias), distancias.shape)  
    bmu = m_pesos[bmu_idx]  
    return bmu, bmu_idx
```

Ilustración 7 Función de calcular BMU

$$D_j = \|X - W_j\| = \sqrt{\sum_{i=1}^N (x_i - w_{ij})^2}$$

Ecuación 1 Distancia Euclídea

La neurona que tenga la menor distancia euclídea respecto al patrón de entrada resultará la ganadora, en código tras realizar este cálculo guardamos sus coordenadas, así como el patrón de la neurona ganadora. La neurona que tenga la menor distancia euclídea respecto al patrón de entrada resultará la ganadora, en código tras realizar este cálculo guardamos sus coordenadas, así como el patrón de la neurona ganadora.

En dicha función se calculará la distancia euclídea de cada neurona respecto al patrón de entrada, para poder realizar esto debemos realizar la raíz cuadrada del sumatorio empezando en 1 hasta la N características del patrón, del i-ésimo elemento del patrón de entrada menos la i-ésima componente de la neurona j, es decir, restar el correspondiente elemento del patrón de entrada con su correspondiente elemento de la matriz de peso, elevándolo al cuadrado para para posteriormente realizar la raíz.

La neurona que tenga la menor distancia euclídea respecto al patrón de entrada resultará la ganadora, en código tras realizar este cálculo guardamos las coordenadas de la ganadora, así como su patrón correspondiente.

```
def variacion_learning_rate(lr_inicial, i, n_iteraciones):  
    eta = lr_inicial * (1 - (i / n_iteraciones))  
    return eta  
✓ 0.0s
```

Ilustración 8 Función para la variación del learning rate



En esta función se calcula el descenso del coeficiente de aprendizaje para cada i-ésima presentación, tomando como referencia su fórmula. Este se debe calcular pues decae linealmente a medida que progresa el entrenamiento, siendo el inicial la primera presentación y 0 en la presentación final.

$$\eta(t) = \eta_0 \left( 1 - \frac{t}{p} \right)$$

Ecuación 2 Variación de eta

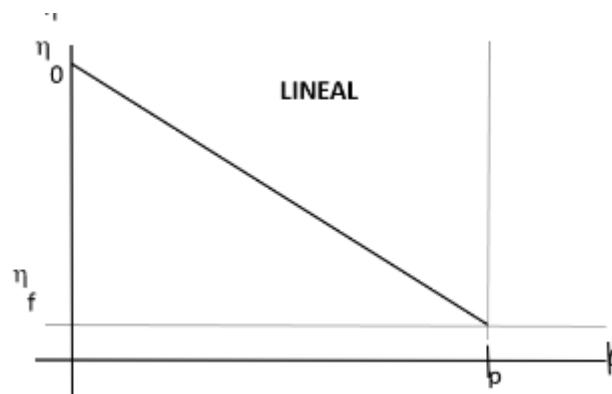


Ilustración 10: Variación lineal de la eta

```
def variacion_vecindario(vecindario_inicial, i, n_iteraciones):
    variacion = 1 + vecindario_inicial * (1 - (i / n_iteraciones))
    return variacion
```

✓ 0.0s

Ilustración 9 Función de variación de vecindario

En esta función calculamos la variación del vecindario que se presentaría en cada iteración, de acorde a la formula, esta debe ser 1 + el vecindario inicial en la presentación 0 y 1 para la presentación final.

$$v(t) = 1 + v_0 \left( 1 - \frac{t}{p} \right)$$

Ecuación 3 Variación de vecindario

```
def decay(distancia_BMU, vecindario_actual):
    return np.exp(-distancia_BMU ** 2 / (2 * vecindario_actual ** 2))
```

✓ 0.0s

Ilustración 11 Función de amortiguación

En esta función debemos calcular la amortiguación acorde a su fórmula, la amortiguación indicará cuanto aprenderá las neuronas vecinas a la BMU

$$a_j = e^{-\frac{d_j^2}{2v_t^2}}$$

Ecuación 4 Amortiguación



```
def calcular_mse(datos, matriz_pesos, patron):  
    """  
    Esta función calcula el error cuadrático medio entre los datos y los pesos de la red SOM.  
    :param datos: matriz de datos de entrada  
    :param matriz_pesos: matriz de pesos de la red SOM  
    :return: valor del error cuadrático medio  
    """  
    # Calcula la BMU  
    mse = 0  
    bmu_aux, bmu_idx_aux = calcular_bmu(patron, matriz_pesos, num_entradas)  
    # Calcula la distancia entre el patrón y la BMU  
    distancia = np.sqrt(np.sum((patron - matriz_pesos) ** 2))  
    mse += distancia ** 2  
    return mse / datos.shape[1]
```

Ilustración 12: Función Calcular MSE

Se ha creado una función para calcular la curva de aprendizaje de la IA, para posteriormente, dibujar la gráfica que muestra cómo aprende la IA y así poder analizarla.

```
def pintar_mapa(matriz_valores, cont):  
    fig = plt.figure()  
  
    # Establece ejes  
    ax = fig.add_subplot(111, aspect='equal')  
    ax.set_xlim((0, matriz_pesos.shape[0]+1))  
    ax.set_ylim((0, matriz_pesos.shape[1]+1))  
    ax.set_title('Self-Organising Map después de %d iteraciones' % periodo)  
  
    # Dibuja los rectángulos de color RGB  
    for x in range(1, matriz_valores.shape[0] + 1):  
        for y in range(1, matriz_valores.shape[1] + 1):  
            ax.add_patch(patches.Rectangle((x-0.5, y-0.5), 1, 1,  
                                           facecolor = matriz_valores[x-1,y-1, :],  
                                           edgecolor = 'none'))  
  
    plt.savefig(f'imagen{cont}.png')  
    plt.show()
```

Ilustración 13 Función pintarmapa()

Esta función permite representar la matriz de pesos en RGB, esta debe estar rellena con números entre 0 y 1, por esto se normaliza.

Además, se crea una imagen .png para después generar un vídeo de cómo varía la matriz de pesos. Como la siguiente

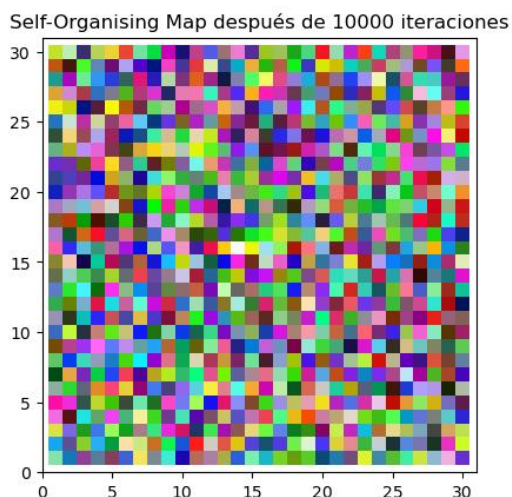


Ilustración 14 Ejemplo de salida de pintarmapa()



```
# Crea la animación
def animacion(cont):
    with imageio.get_writer('animacion.mp4', fps=30) as writer:
        for i in range(cont):
            filename = f'imagen{i}.png'
            image = imageio.imread(filename)
            writer.append_data(image)
```

✓ 0.0s

Ilustración 15 Función de animación

Esta función genera un vídeo en MP4 juntando todas las imágenes de generadas en *pintarmapa()*. Para esta función hemos hecho uso del chatbot gpt3. Además, es necesario instalar la biblioteca *imageio*. Introduciendo por consola el siguiente código:

```
pip install imageio
```

Ilustración 16 Instalación de imageio

Para pintar la curva de aprendizaje y la variación del *learning rate* se hace uso de las siguientes funciones:

```
def plot_learning_rate(lrate):
    """
    Esta función grafica la variación del learning rate en cada iteración.
    :param lrate: una lista de valores de learning rate en cada iteración
    """
    # Crear la figura
    fig, ax = plt.subplots()

    # Crear el gráfico de línea
    ax.plot(lrate, label='Learning rate')

    # Añadir etiquetas y título
    ax.set_xlabel('Iteración')
    ax.set_ylabel('Valor del learning rate')
    ax.set_title('Variación del learning rate')

    # Mostrar la leyenda y el gráfico
    ax.legend()
    plt.show()
```

Ilustración 17 Función para pintar la variación del learning rate



```
def plot_curva_aprendizaje(mse):  
    """  
    Función que grafica la curva de aprendizaje de un SOM.  
  
    Args:  
    mse: ndarray, array que contiene el error cuadrático medio por época.  
    """  
    plt.plot(range(len(mse)), mse)  
    plt.title('Curva de aprendizaje')  
    plt.xlabel('Época')  
    plt.ylabel('Error cuadrático medio')  
    plt.show()
```

Ilustración 18 Función para pintar la curva de aprendizaje

### 3.2.3 Entrenamiento

En esta parte del código se ha implementado el entrenamiento de la red SOM. Para ello se escoge un patrón del set de datos que se creó al azar y se compara con la matriz de pesos. Se ha decidido hacer dos bucles para poder asegurar que se escogen todos los patrones de datos y se hacen todas las iteraciones solicitadas por teclado.

Además, se ha decidido complementar esta función con una nueva, generada por el *chatbot* GPT3 para mostrar la variación de los learning rate al finalizar el entrenamiento.

Lo primero en calcular es la neurona BMU, calcular BMU permite guardar, tanto el vector de pesos de la matriz ganadora como sus coordenadas. De esta manera las guardamos en dos variables distintas para poder usarlas a continuación.

En segundo lugar, se debe calcular la variación del *learning rate* y la variación del vecindario, guardamos la variación del vecindario.

Por último, se actualiza la matriz de pesos con los datos obtenidos anteriormente. Para ello, se hacen dos bucles para recorrer la matriz de pesos de forma que se puede calcular la distancia entre la matriz de pesos y la ganadora. Si la distancia es menor o igual al tamaño del vecindario. Se calcula la amortiguación y las neuronas del vecindario se actualizan y recompensan, pero no tanto como la ganadora.

Cada 100 iteraciones, se muestra cómo se va modificando la matriz de pesos, gracias a la función *pintarmapa()*. Además, se guardan las imágenes para posteriormente crear el vídeo de la evolución, como ya se ha comentado anteriormente.

Cuando termina el entrenamiento se muestra el mapa completo y una gráfica con la variación del learning rate.



```
# Entrena la red con el dataset de entrenamiento
cont = 0
mse = []
if os.path.exists("Evolucion_julian") == 0:
    os.mkdir("Evolucion_julian")
etahistoricos = np.zeros(periodo)
pintar_mapa(matriz_pesos, cont)
for i in range(periodo):
    for patron in datos.T:
        # Calcula la BMU
        bmu, bmu_idx = calcular_bmu(patron, matriz_pesos, num_entradas)
        # Calcula el descenso del coeficiente de aprendizaje y del vecindario
        eta = variacion_learning_rate(learning_rate, i, periodo)
        etahistoricos[i] = eta
        v = variacion_vecindario(vecindario, i, periodo)
        # Actualiza los pesos de las neuronas
        for x in range(matriz_pesos.shape[0]):
            for y in range(matriz_pesos.shape[1]):
                w = matriz_pesos[x, y, :]
                distancia = np.sqrt(np.sum((np.array([x, y]) - np.array(bmu_idx)) ** 2))
                if distancia <= v:
                    # Calcula la amortiguación de eta
                    amortiguacion = decay(distancia, v)
                    # Actualiza los pesos
                    matriz_pesos[x, y, :] = w + (eta * amortiguacion * (patron - w))
            mse.append(calcular_mse(datos, matriz_pesos, patron))
        if i % 100 == 0:
            print('Iteración %d/%d' % (i, periodo))
            cont += 1
            pintar_mapa(matriz_pesos, cont)

print('Entrenamiento finalizado')
pintar_mapa(matriz_pesos, cont)
plot_curva_aprendizaje(mse)
plot_learning_rate(etahistoricos)
```

Ilustración 18 Código de aprendizaje de la IA

### 3.2.4 Clasificación

En el siguiente bloque de código clasificamos el dataset de patrones con la matriz de pesos entrenada. Para ello inicializamos tres matrices, mapa de clasificación, mapa de activaciones y mapa de distancias.

```
# Clasifica los patrones de entrenamiento con la matriz de pesos recién entrenada
# Definir y inicializar las matrices
mapa_clasificacion = np.zeros(matriz_pesos.shape)
mapa_activaciones = np.zeros((lado_mapa, lado_mapa))
mapa_distancias = np.zeros((lado_mapa, lado_mapa))

# Recorrer todo el dataset de entrenamiento
for patron_entrada in datos.T: # Obtener el patrón de entrada
    # Calcular la BMU y sus coordenadas
    bmu, bmu_idx = calcular_bmu(patron_entrada, matriz_pesos, num_entradas)
    bmu_x, bmu_y = bmu_idx

    # Imprimir las coordenadas de la BMU
    print("Patrón {}: BMU en ({}, {})".format(patron_entrada, bmu_x, bmu_y))

    # Actualizar los mapas de clasificación, activaciones y distancias
    mapa_clasificacion[bmu_x, bmu_y] = patron_entrada
    mapa_activaciones[bmu_x, bmu_y] += 1

    #mapa_activaciones[bmu_x, bmu_y] = num_activ
    mapa_distancias[bmu_x, bmu_y] += np.linalg.norm(patron_entrada - matriz_pesos[bmu_x, bmu_y])
    mapa_distancias[bmu_x, bmu_y] += np.mean(np.linalg.norm(patron_entrada - matriz_pesos[bmu_x, bmu_y]))

# Calcular el número de clases
num_clases = len(np.unique(mapa_clasificacion))
print("Numero de clases ", num_clases)

# Calcular la distancia media y el Error Topológico del mapa
error_cuantificacion = np.mean(np.linalg.norm(matriz_pesos - mapa_clasificacion, axis=2)) / num_entradas
print("Error_cuantificacion ", error_cuantificacion)

error_topologico = 0
for x in range(lado_mapa):
    for y in range(lado_mapa):
        vecinos = [(i, j) for i in range(x-1, x+2) for j in range(y-1, y+2) if (i >= 0 and i < lado_mapa and j >= 0 and j < lado_mapa)]
        vecinos_activos = [v for v in vecinos if mapa_activaciones[v] > 0]
        if vecinos_activos:
            vecinos_distancias = [np.linalg.norm(matriz_pesos[x, y] - matriz_pesos[i, j]) for (i, j) in vecinos_activos]
max_distancia = np.nanmax(vecinos_distancias)
min_distancia = np.nanmin(vecinos_distancias)
if min_distancia > 0:
    error_topologico += max_distancia / min_distancia
```

Ilustración 19 Código de clasificación



Una vez tenemos la matriz entrenada vamos introduciendo los respectivos valores a cada una de las matrices pedidas:

**Mapa de clasificación:** matriz de iguales dimensiones que la matriz de pesos para guardar en cada neurona el último patrón clasificado, esto es realizado en el código almacenando en la posición de la bmu el patrón de entrada.

```
mapa_clasificacion[bmu_x, bmu_y] = patron_entrada
```

*Ilustración 20: código mapa clasificación*

**Mapa de activaciones:** matriz bidimensional para guardar el número de patrones reconocido por cada neurona, en el mapa de activaciones sumaremos uno a la variable correspondiente en la posición de la neurona.

```
mapa_activaciones[bmu_x, bmu_y] += 1
```

*Ilustración 21 código mapa activación*

**Mapa de distancias:** matriz bidimensional para guardar, para las neuronas con activación>0, la distancia media de todos los patrones de la clase con su vector de pesos. Esto es realizado en el código calculando la distancia euclídea media, acorde a su fórmula.

```
mapa_distancias[bmu_x, bmu_y] += np.mean(np.linalg.norm(patron_entrada - matriz_pesos[bmu_x, bmu_y]))
```

*Ilustración 22 código mapa distancias*

$$D_j = \|X - W_j\| = \sqrt{\sum_{i=1}^N (x_i - w_{ij})^2}$$

*Ecuación 5 Distancia Euclídea*

### 3.2.5 Prueba

En este último SOM buscamos realizar el último apartado de la práctica que posteriormente explicamos, incluyendo varios segmentos para su desempeño, lo colocamos aquí pues se puede visualizar el código de manera más sencilla y limpia.





## SOM Prueba

```
# Clasifica nuevos patrones
mapa_clasificacion_prueba = np.zeros(matriz_pesos.shape)
lista_colores = [[255, 255, 255], [255, 0, 0], [0, 255, 0], [0, 0, 255], [255, 255, 0], [255, 0, 255], [0, 255, 255], [0, 0, 0]]
lista_colores = np.divide((lista_colores), (max(lista_colores)))

print("BMU Activadas")
# Recorrer todo el dataset de prueba
for patron_entrada in lista_colores:
    # Calcular la BMU y sus coordenadas
    bmu, bmu_idx = calcular_bmu(patron_entrada, matriz_pesos, num_entradas)
    bmu_x, bmu_y = bmu_idx

    # Actualizar los mapas de clasificación, activaciones y distancias
    mapa_clasificacion_prueba[bmu_x, bmu_y] = patron_entrada
    mapa_activaciones[bmu_x, bmu_y] += 1
    mapa_distancias[bmu_x, bmu_y] += np.mean(np.linalg.norm(patron_entrada - matriz_pesos[bmu_x, bmu_y]))

    print(bmu_idx)

# Calcular el número de clases en el dataset de prueba
num_clases_prueba = len(np.unique(mapa_clasificacion_prueba))

# Calcular el error de cuantificación en el dataset de prueba
error_cuantificacion_prueba = np.mean(np.linalg.norm(matriz_pesos - mapa_clasificacion_prueba, axis=2)) / num_entradas

# Calcular el error topológico en el dataset de prueba
error_topologico_prueba = 0
for x in range(lado_mapa):
    for y in range(lado_mapa):
        vecinos = [(i, j) for i in range(x-1, x+2) for j in range(y-1, y+2) if (i >= 0 and i < lado_mapa and j >= 0 and j < lado_mapa)]
        vecinos_activos = [v for v in vecinos if mapa_activaciones[v] > 0]
        if vecinos_activos:
            vecinos_distancias = [np.linalg.norm(matriz_pesos[x, y] - matriz_pesos[i, j]) for (i, j) in vecinos_activos]
            max_distancia = np.nanmax(vecinos_distancias)
            min_distancia = np.nanmin(vecinos_distancias)
            if min_distancia > 0:
                error_topologico += max_distancia / min_distancia
```

Ilustración 23: Código Som de Prueba



### 3.3 Cuestiones

Se van a responder varias cuestiones a partir de la práctica realizada.

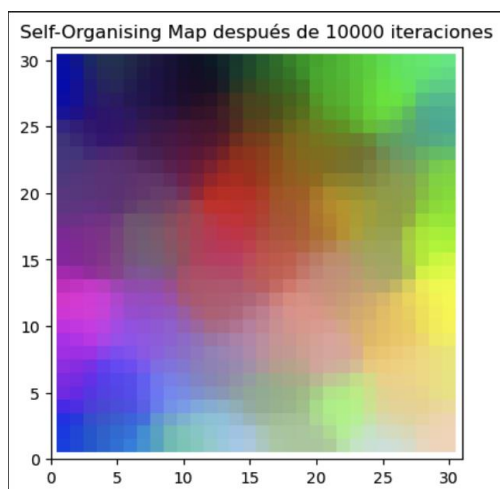
#### 3.3.1 Cuestión 1

***“¿Cuáles son los valores de Lado\_Mapa, Periodo y Eta más adecuados? Realiza gráficas como las vistas en teoría para justificar tu selección de los valores concretos de estos parámetros y explica el motivo de tu elección”***

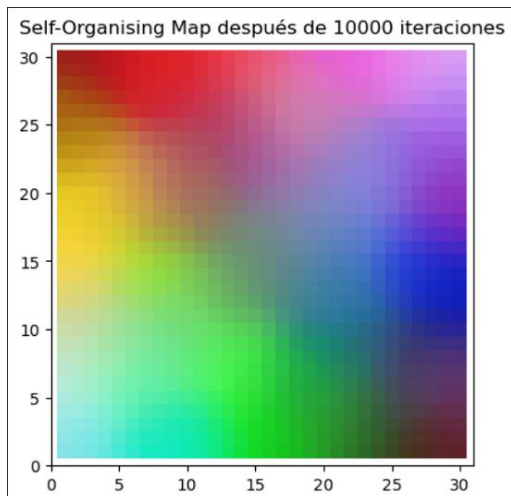
Respecto al lado del mapa, tras muchos entrenamientos y pruebas realizadas, descubrimos que **30 es el lado más adecuado**, esto es debido a que la red necesita suficientes colores para poder realizar una clasificación correcta pero no excesivos para evitar un sobre entrenamiento y tener un coste de tiempo de entrenamiento muy alto, investigando nos percatamos que, por debajo de 30, es decir con 20 colores no contiene los datos suficientes y con más de 30 es demasiado costoso.

[Video](#) del proceso de aprendizaje

De las múltiples combinaciones realizadas mostramos algunas comparaciones con parámetros parecidos. Aquí podemos observar una red entrenada con 30 de lado mapa, 10000 interacciones y 0.01 de eta (izquierda), una red entrenada con 30 de lado mapa, 10000 interacciones y 0.001 de eta (derecha) y por último la red seleccionada, entrenada con 30 de lado mapa, 10000 interacciones y 0.0001 de eta.



*Ilustración 25: RGB de pesos tras 3000 interacciones con 0.005 eta y 30 de lado*



*Ilustración 24: RGB de pesos tras 10000 interacciones con 0.0001 eta y 30 de lado*

En el marco del periodo, este va subordinado con el Eta pues en cada periodo los pesos de la red se ajustarán. Cuanto mayor sea la eta, mayor será el ajuste que se realice en cada neurona de la red y la propia red convergerá más rápidamente. En cambio, si la eta es muy bajo el ajuste será menor y la red convergerá muy lentamente, pero será más efectiva. Por lo tanto, para un buen aprendizaje, la eta y el periodo deben ser indirectamente proporcionales.

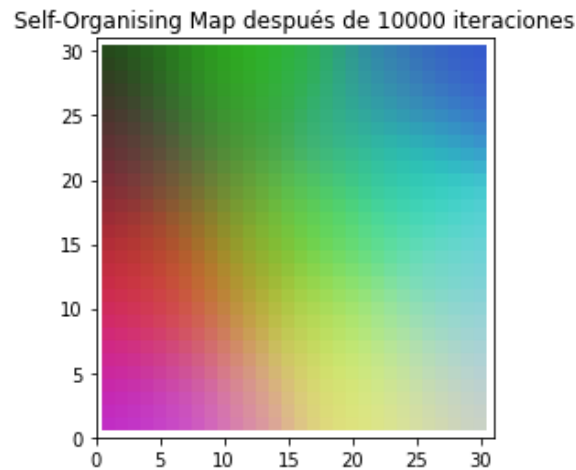


Ilustración 26 Entrenamiento con 30 de lado, 10000 iteraciones y 0.0001 eta

Con todo esto, podemos afirmar que, con un lado de mapa de 30, **un periodo de 10000 interacciones** y una eta de 0.0001 obtenemos muy buenos resultados y la clasificación se ve realmente bien. Como se observa en las siguientes ilustraciones:

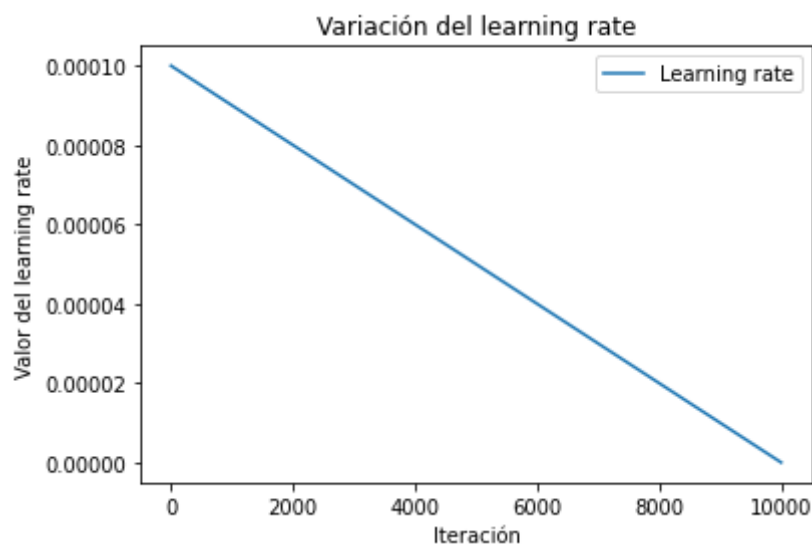


Ilustración 27: Variación del aprendizaje a lo largo del entrenamiento

Con todo lo mencionado anteriormente se concluye **que 0.0001 de eta es el valor más adecuado** para poder un aprendizaje correcto, pues en un entrenamiento posterior con un valor más bajo de 0.00001 el aprendizaje era demasiado lento y costoso y poco eficiente pues era un aprendizaje mínimo entre interacción.

Aquí podemos observar la curva de aprendizaje de una red entrenada con 30 de lado mapa, 10000 interacciones y 0.01 de eta (3ª imagen), una red entrenada con 30 de lado mapa, 10000 interacciones y 0.001 de eta (2ª imagen) y por último la red seleccionada, entrenada con 30 de lado mapa, 10000 interacciones y 0.0001 de eta (1ª imagen).

Como se observa a medida que el coeficiente de aprendizaje es menor mas estable es la curva de aprendizaje.

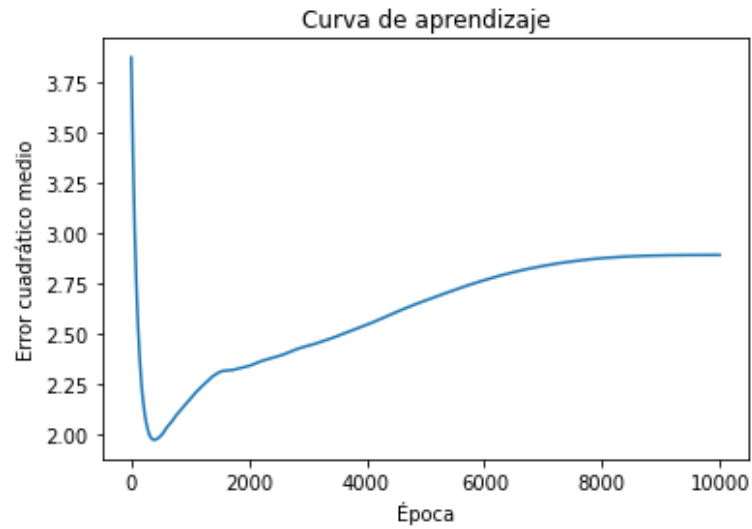


Ilustración 30 Curva de aprendizaje con 10000 iteraciones, 30 de lado y eta 0,0001

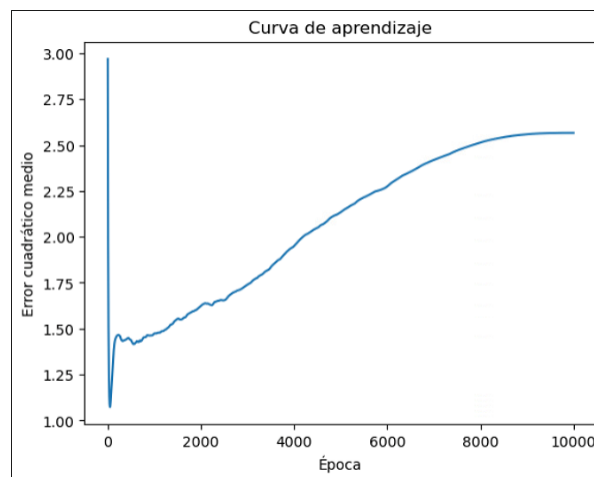


Ilustración 28 Curva de aprendizaje con 10000 iteraciones, 30 de lado y eta 0,001

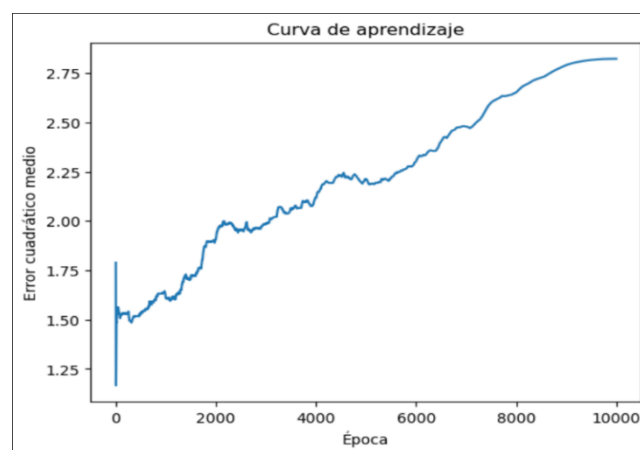


Ilustración 29 Curva de aprendizaje con 10000 iteraciones, 30 de lado y eta 0,01



### 3.3.2 Cuestión 2

***“Para mejor clasificación que hayas obtenido del dataset de entrenamiento, adjunta la información generada en la ejecución y explica detalladamente que estamos viendo en cada uno de ellos. Deberás de entregar***

***2.1. Gráfico RGB de los pesos iniciales.***

***2.2. Gráfico RGB de los pesos entrenados.***

***2.3 Número de clases, mapa de clasificación (gráfico RGB), mapa de activaciones (histograma 3D), mapa de distancias del dataset, el error de cuantificación y el error topológico “***

Gráfico RGB de los pesos iniciales: generados aleatoriamente

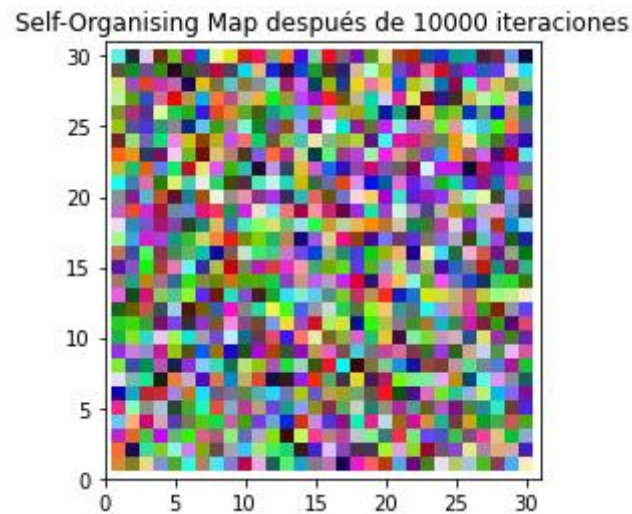


Ilustración 31 Matriz de pesos para  $t(0)$

Gráfico RGB de los pesos entrenados:

- Numero de clases:
  - Aquí observamos que tras la clasificación se forman un total de 169 clases del conjunto total.

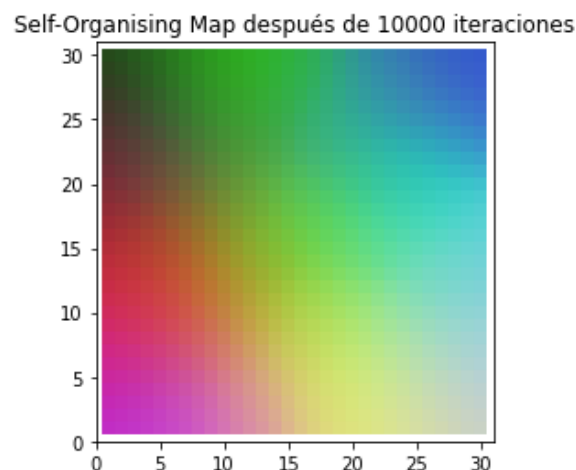


Ilustración 32 Matriz de pesos para  $t(10000)$

Como se puede ver en el notebook, el número de clases es: 169.



- Mapa de clasificación:
  - Aquí representamos las diferentes clases clasificadas en el entrenamiento.

Self-Organising Map después de 10000 iteraciones

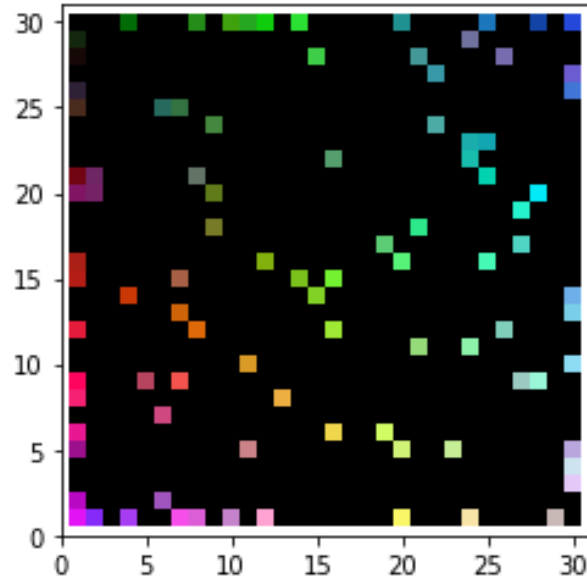


Ilustración 33 Mapa de clasificación tras el entrenamiento

- Mapa de activaciones:
  - Aquí se indica las neuronas que se han activado y la cantidad de veces que se ha activado cada uno, viendo que como máximo se activa 4 veces una neurona.

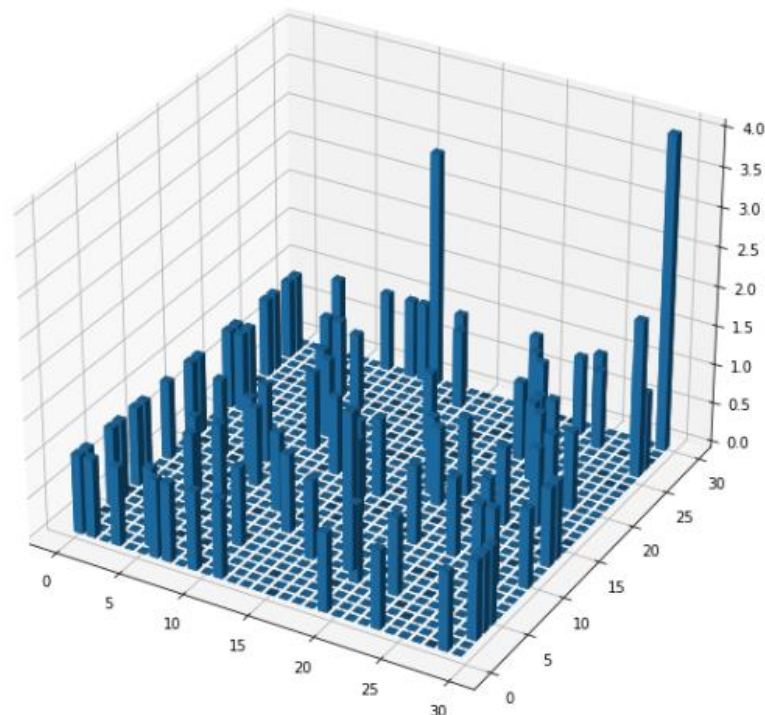


Ilustración 34 Mapa de activaciones al final del entrenamiento



- Mapa de distancias:
  - Debido al tamaño de la matriz, se debe de visualizar en el cuaderno entregado.
- Error de cuantificación:
  - El error de cuantificación es una medida del grado de aproximación de una unidad (neurona) del mapa auto-organizado (SOM) a su correspondiente vector de entrada en el espacio de características. En este caso podemos observar que es un numero bajo por lo que es una aproximación cercana al patrón. Como se muestra en el Notebook, el error de cuantificación es: 0.29369190961957465.
- Error topológico:
  - El error topológico se refiere a la discrepancia entre la topología del mapa y la topología de los datos, en este caso podemos observar que al dar el valor 0, no existe discrepancia.

### 3.3.3 Cuestión 3

***“Sin modificar el entrenamiento, clasifica el dataset de prueba y adjunta los mismos resultados que en el apartado 2.3. Explícalos [255, 255, 255] [ 255, 0, 0] [0, 255, 0] [0, 0, 255] [ 255, 255, 0] [255, 0, 255] [0, 255, 255] [0, 0, 0].”***

Para realizar esto creamos una lista con los colores pedidos en el enunciado, lo primero que realizamos es normalizar la lista con los patrones, esto se realiza dividiendo los elementos de la lista entre el máximo existente, para poder realizar el posterior entrenamiento y obtener las conclusiones.

```
# Clasifica nuevos patrones
mapa_clasificacion_prueba = np.zeros(matriz_pesos.shape)
lista_colores = [[255, 255, 255], [255, 0, 0], [0, 255, 0], [0, 0, 255], [255, 255, 0], [255, 0, 255], [0, 255, 255], [0, 0, 0]]
lista_colores = np.divide((lista_colores), (max(lista_colores)))
```

Ilustración 35: Código colores de entrenamiento declaración

Al igual que en anteriores apartados programamos el SOM para guardar los datos correspondientes al apartado y posteriormente imprimirlos por pantalla. Pintando la posición de cada BMU en la red.



Posteriormente dibujamos los mapas, el número de clases y los respectivos errores:

```
# Recorrer todo el dataset de prueba
for patron_entrada in lista_colores:
    # Calcular la BMU y sus coordenadas
    bmu, bmu_idx = calcular_bmu(patron_entrada, matriz_pesos, num_entradas)
    bmu_x, bmu_y = bmu_idx

    # Actualizar los mapas de clasificación, activaciones y distancias
    mapa_clasificacion_prueba[bmu_x, bmu_y] = patron_entrada
    mapa_activaciones[bmu_x, bmu_y] += 1
    mapa_distancias[bmu_x, bmu_y] += np.mean(np.linalg.norm(patron_entrada - matriz_pesos[bmu_x, bmu_y]))
    print(bmu_idx)

# Calcular el número de clases en el dataset de prueba
num_clases_prueba = len(np.unique(mapa_clasificacion_prueba))
# Calcular el error de cuantificación en el dataset de prueba
error_cuantificacion_prueba = np.mean(np.linalg.norm(matriz_pesos - mapa_clasificacion_prueba, axis=2)) / num_entradas

# Calcular el error topológico en el dataset de prueba
error_topologico_prueba = 0
for x in range(lado_mapa):
    for y in range(lado_mapa):
        vecinos = [(i, j) for i in range(x-1, x+2) for j in range(y-1, y+2) if (i >= 0 and i < lado_mapa and j >= 0 and j < lado_mapa)]
        vecinos_activos = [v for v in vecinos if mapa_activaciones[v] > 0]
        if vecinos_activos:
            vecinos_distancias = [np.linalg.norm(matriz_pesos[x, y] - matriz_pesos[i, j]) for (i, j) in vecinos_activos]
            max_distancia = np.nanmax(vecinos_distancias)
            min_distancia = np.nanmin(vecinos_distancias)
            if min_distancia > 0:
                error_topologico_prueba += max_distancia / min_distancia
```

Ilustración 36 Código colores de cuestión 3 entrenamiento

Por último, mostramos los mapas que se solicitan.

```
pintar_mapa(mapa_clasificacion_prueba)

fig = plt.figure(figsize = (10,10))
hist = fig.add_subplot(111, projection='3d')

xpos = [range(mapa_activaciones.shape[0])] # El largo del eje x es igual a una lista que guarda el número de neuronas en el mapa de activación
ypos = [range(mapa_activaciones.shape[0])]
#print(ypos)
xpos, ypos = np.meshgrid(xpos, ypos) # Crea una matriz a partir de las listas creadas con anterioridad
#print(ypos)
xpos = xpos.flatten('F') # Pasa la matriz creada a una lista, la f indica que se hace por columnas de la matriz
ypos = ypos.flatten('F')
#print(ypos)
zpos = np.zeros_like(xpos) # Creamos una lista del mismo tamaño que x que rellenaremos con los valores del mapa

dx = dy = 0.7 * np.ones_like(zpos) # Indicamos el grosor de las barras

dz = mapa_activaciones.flatten() # Damos el valor a la z con el del mapa de activaciones

#print(dz)

hist.bar3d(xpos, ypos, zpos, dx, dy, dz, zsort='average')
plt.show()

print(mapa_distancias)
print("Numero de clases en dataset de prueba: ", num_clases_prueba)
print("Error de cuantificación en dataset de prueba: ", error_cuantificacion_prueba)
print("Error de topologico en dataset de prueba: ", error_topologico_prueba)
```

Ilustración 37 Mapas de clasificación, activaciones, distancias y número de clases, error de cuantificación y error topológico de las pruebas

- Numero de clases:
  - Aquí observamos el número de clases nuevas que se forman. Como podemos ver en el Notebook, el número de clases es 2.





- Mapa de clasificación:
  - Aquí representamos las neuronas que se han activado al introducir los datos

Self-Organising Map después de 10000 iteraciones

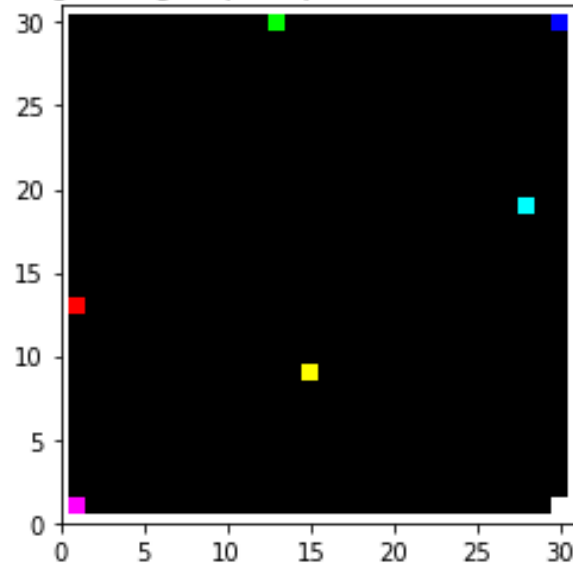


Ilustración 38 Mapa de clasificación de la prueba

- Mapa de activaciones:
  - Aquí se indica las neuronas que se han activado y la cantidad de veces que se ha activado cada uno, viendo que como máximo se activa 5 veces una neurona.

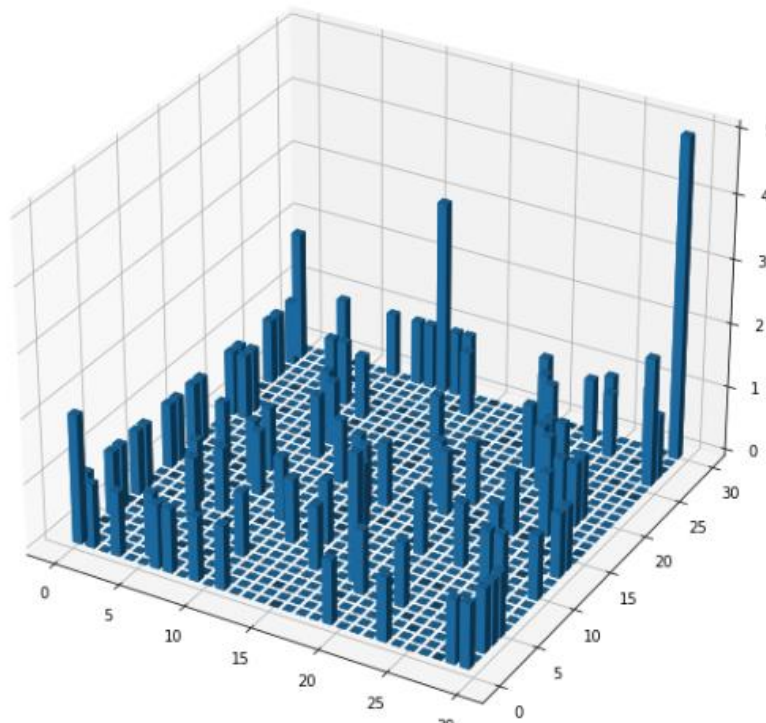


Ilustración 39 Histograma 3D de activación de la prueba



- Mapa de distancias:
  - Debido al tamaño de la matriz, se debe de visualizar en el cuaderno entregado.
- Error de cuantificación:
  - El error de cuantificación es una medida del grado de aproximación de una unidad (neurona) del mapa autoorganizado (SOM) a su correspondiente vector de entrada en el espacio de características. En este caso podemos observar que pese a ser un número superior al obtenido durante el entrenamiento, continúa siendo un numero bajo por lo que es una aproximación cercana al patrón.
  - Como podemos ver en el Notebook, el error de cuantificación es 0.32003844968055456.
- Error topológico:
  - El error topológico se refiere a la discrepancia entre la topología del mapa y la topología de los datos, en este caso podemos observar que al dar el valor 0, no existe discrepancia.

### 3.4- Conclusiones

En conclusión, la utilización de un SOM (Mapas autoorganizativos) en la clasificación de colores es una técnica muy útil y efectiva para agrupar grandes conjuntos de datos y reducir su complejidad.

Al entrenar el SOM con una gran cantidad de datos de entrada, se puede lograr que el mapa tenga una estructura que refleje las características de los datos, lo que permite agrupar los colores en diferentes categorías en función de su similitud y permite la visualización de los resultados de una manera intuitiva y fácil de interpretar. Al utilizar un mapa de colores para representar los grupos o categorías, se puede identificar rápidamente la distribución de los colores y su similitud en el espacio de característica.



## 4.- Práctica 2

### 4.1- Introducción

En esta práctica se busca aplicar el algoritmo de Mapas Autoorganizados (SOM), realizado en la práctica anterior, a un conjunto de datos de 171 países del mundo correspondientes a mediciones del año 2010 en 17 características diferentes, con el objetivo de agrupar a los países con características similares.

Las características incluyen indicadores socioeconómicos como el PIB per cápita, la tasa de mortalidad infantil y la tasa de fecundidad adolescente, entre otros. Para ello, se utilizará el lenguaje de programación Python para entrenar el modelo SOM. Posteriormente, se analizarán los mapas resultantes y se extraerán conclusiones sobre los patrones encontrados.

### 4.2- Implementación

Debido a la similitud de objetivos en ambas partes de la práctica se ha decidido documentar en esta parte del documento tan solo las partes que no son iguales a lo ya comentado para no sobrecargar el documento.

Comenzamos importando las librerías necesarias y leyendo el archivo “Countries2010.csv” para pasarlo a un *dataframe* que posteriormente manipularemos.

```
# from __future__ import division
import os
import json
import numpy as np
from matplotlib import pyplot as plt
from matplotlib import patches as patches
import pandas as pd
from mpl_toolkits.basemap import Basemap
import tensorflow as tf
from tensorflow import keras
from keras.models import load_model
import geopandas as gpd

%matplotlib inline
```

Ilustración 40 Importación de librerías

Se guardan los datos del fichero CSV para, posteriormente, almacenarlos en un *dataframe* llamado *datos*.

```
# Código para obtener el Dataset que se va a usar en el entrenamiento
datos = pd.read_csv("Countries2010.csv")
```

Ilustración 41 Lectura del CSV y guardado de datos



#### 4.2.1 Entrenamiento

Utilizando como base el SOM de entrenamiento de la práctica 1, hemos realizados diferentes modificaciones para poder adaptarlo a un dataset con 17 elementos a analizar, las modificaciones han sido las siguientes:

- Eliminado el contador para pintar el mapa correspondiente, para ser pintado posteriormente.
- Añadido la variable *aux* que contendrá todos los valores correspondientes a cada país que posteriormente serán pasados como patrón para calcular la BMU.

```
# Entrena la red con el dataset de entrenamiento
mse = []
etahistoricos = np.zeros(periodo)
for i in range(periodo):
    for patron in normalized_data.T:
        # Calcula la BMU
        aux = normalized_data.T[patron].iloc[1:].values

        bmu, bmu_idx = calcular_bmu(aux, matriz_pesos, num_entradas)
        # Calcula el descenso del coeficiente de aprendizaje y del vecindario
        eta = variacion_learning_rate(learning_rate, i, periodo)
        etahistoricos[i] = eta
        v = variacion_vecindario(vecindario, i, periodo)
        # Actualiza los pesos de las neuronas
        for x in range(matriz_pesos.shape[0]):
            for y in range(matriz_pesos.shape[1]):
                w = matriz_pesos[x, y, :]

                distancia = np.sqrt(np.sum((np.array([x, y]) - np.array(bmu_idx)) ** 2))
                if distancia <= v:
                    # Calcula la amortiguación de eta
                    amortiguacion = decay(distancia, v)
                    # Actualiza los pesos
                    matriz_pesos[x, y, :] = w + (eta * amortiguacion * (aux - w))
        mse.append(calcular_mse(normalized_data, matriz_pesos, aux))

    if i % 100 == 0:
        print('Iteración %d/%d' % (i, periodo))
print("Entrenamiento finalizado")
```

Ilustración 42 Entrenamiento Práctica 2

#### 4.2.2- Clasificación

Para clasificar usamos como referencia el código de clasificación del laboratorio 1, pero adaptamos la funcionalidad al nuevo problema. Creamos un diccionario para guardar todos los países que son reconocidos por cada neurona, además de otro diccionario en el que guardamos los vectores de pesos de las neuronas que se han activado en tu mapa para las entradas de Venezuela, Noruega y Etiopía.



```
mapa_clasificacion_Modificado = {}
for i in range (lado_mapa):
    for j in range (lado_mapa):
        mapa_clasificacion_Modificado['({}, {})' .format(i,j)] = []

mapa_clasificacion_Con_vectores = {'Ethiopia' : [], 'Venezuela, RB' : [], 'Norway' : []}

mapa_clasificacion = np.zeros(matriz_pesos.shape)
mapa_activaciones = np.zeros((lado_mapa, lado_mapa))
mapa_distancias = np.zeros((lado_mapa, lado_mapa))
print("Mapa de clasificación con la neurona, país y vector. \n")
# Recorrer todo el dataset de entrenamiento
for patron_entrada in normalized_data.T: # Obtener el patrón de entrada
    # Calcular la BMU y sus coordenadas
    aux = normalized_data.T[patron_entrada].iloc[1:].values
    bmu, bmu_idx = calcular_bmu(aux, matriz_pesos, num_entradas)
    bmu_x, bmu_y = bmu_idx
    coord = '(' + str(bmu_x) + ', ' + str(bmu_y) + ')'
    mapa_clasificacion[bmu_x, bmu_y] = patron
    mapa_clasificacion_Modificado[coord].append(datos.CountryName[patron_entrada])
    if(datos.CountryName[patron_entrada] in mapa_clasificacion_Con_vectores):
        mapa_clasificacion_Con_vectores[datos.CountryName[patron_entrada]].append(bmu)
    # Imprimir las coordenadas de la BMU

    print("Nombre del País {}: BMU en ({}, {}) con vector {}".format(datos.CountryName[patron_entrada], bmu_x, bmu_y, bmu))

# Actualizar los mapas de clasificación, activaciones y distancias

mapa_activaciones[bmu_x, bmu_y] += 1

#mapa_activaciones[bmu_x, bmu_y] = num_activ
#mapa_distancias[bmu_x, bmu_y] += np.linalg.norm(patron_entrada - matriz_pesos[bmu_x, bmu_y])
mapa_distancias[bmu_x, bmu_y] += np.mean(np.linalg.norm(aux - matriz_pesos[bmu_x, bmu_y]))
# Calcular el número de clases
num_clases = len(np.unique(mapa_clasificacion))

# Calcular la distancia media y el Error Topológico del mapa
error_cuantificacion = np.mean(np.linalg.norm(matriz_pesos - mapa_clasificacion, axis=2)) / num_entradas

error_topologico = 0
for x in range(lado_mapa):
    for y in range(lado_mapa):
        vecinos = [(i, j) for i in range(x-1, x+2) for j in range(y-1, y+2) if (i >= 0 and i < lado_mapa and j >= 0 and j < lado_mapa)]
        vecinos_activos = [v for v in vecinos if mapa_activaciones[v] > 0]
        if vecinos_activos:
            vecinos_distancias = [np.linalg.norm(matriz_pesos[x, y] - matriz_pesos[i, j]) for (i, j) in vecinos_activos]
max_distancia = np.nanmax(vecinos_distancias)
min_distancia = np.nanmin(vecinos_distancias)
if min_distancia > 0:
    error_topologico += max_distancia / min_distancia
```

Ilustración 43 Clasificación Práctica 2

### 4.3- Cuestiones

Se van a contestar las diferentes funciones de la práctica.

#### 4.3.1 Cuestión 1

***“¿Cuáles son los valores de Lado Mapa, Periodo y Eta más adecuados? Realiza gráficas como las vistas en teoría para justificar tu selección de los valores concretos de estos parámetros y explica el motivo de tu elección “***

Tras realizar muchos entrenamientos con diversos valores y varias combinaciones podemos llegar a la conclusión que los mejores valores son **5000 de periodo, 0.0001 de eta y 20 de lado mapa.**

En los numerosos entrenamientos realizados observamos que tener menos de 5000 interacciones hace que no se pueda clasificar los datos de manera apropiada, hacer uso de menos de 15 de lado del mapa resulta insuficiente y en especial si se hace uso de una eta superior a 0.0001 el aprendizaje resulta inefectivo debido que pese a aprender más rápido se realiza un aprendizaje de peor calidad.

Aquí se puede mostrar el contraste entre la red entrenada con los valores adecuados (1ª imagen) y diversas redes entrenadas con valores similares, pero otorgan peores resultados.



- 1º imagen: una red con los valores seleccionados por ser los más adecuados
- 2º imagen: una red entrenada con 20 de lado, 5000 interacciones y 0.01 de eta.
- 3º imagen: una red entrenada con 15 de lado, 2500 interacciones y 0.01 de eta.
- 4º imagen: una red entrenada con 15 de lado, 2500 interacciones y 0.001 de eta.
- 5º imagen: una red entrenada con 20 de lado, 2500 interacciones y 0.1 de eta.

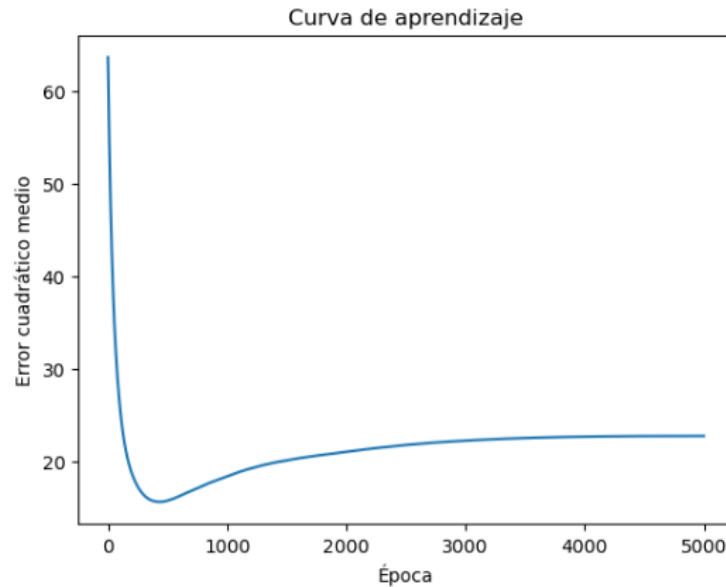


Ilustración 44 Curva de aprendizaje con Lado de mapa 20, iteraciones 5000 y eta 0.0001

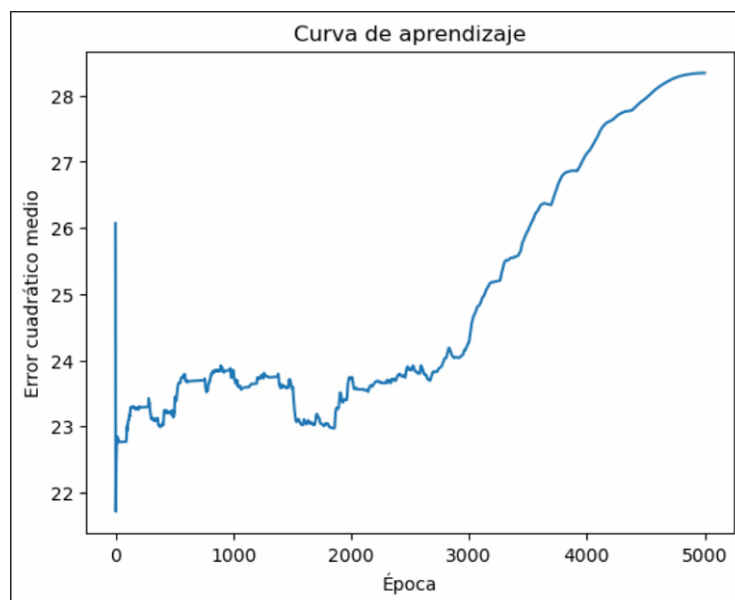


Ilustración 45 Curva de aprendizaje con Lado de mapa 20, 5000 iteraciones y 0.001 eta

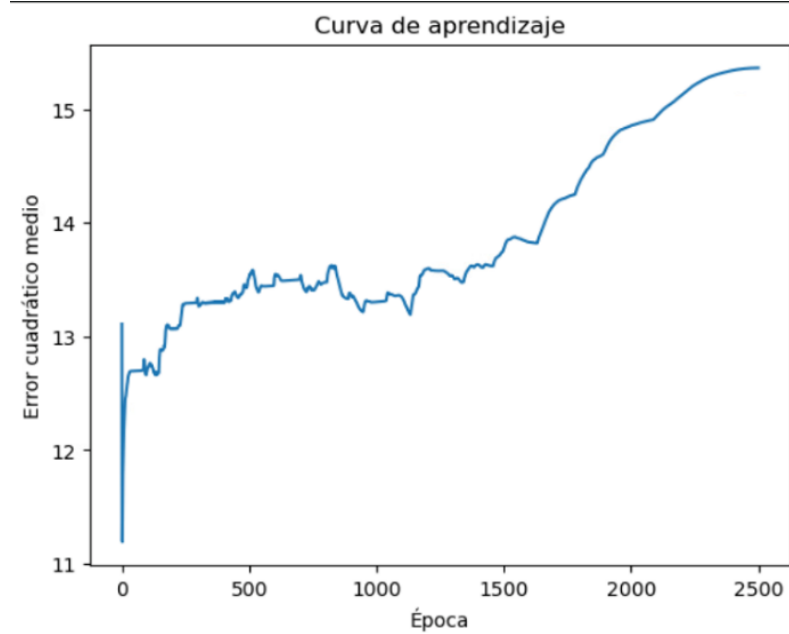


Ilustración 46 Curva de aprendizaje con Lado de mapa 15, 2500 iteraciones y 0.01 de eta

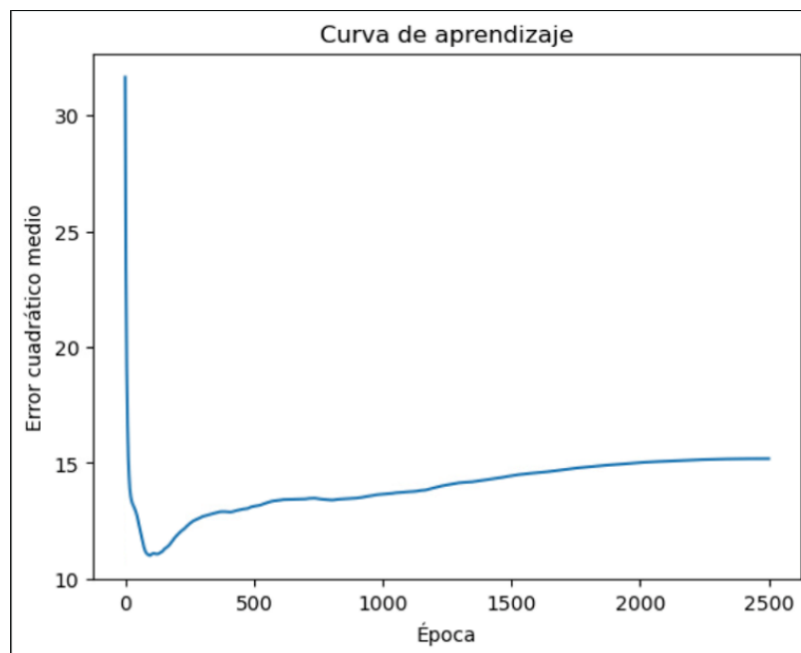


Ilustración 47 Curva de aprendizaje con Lado de mapa 15, 2500 iteraciones y 0.001 eta

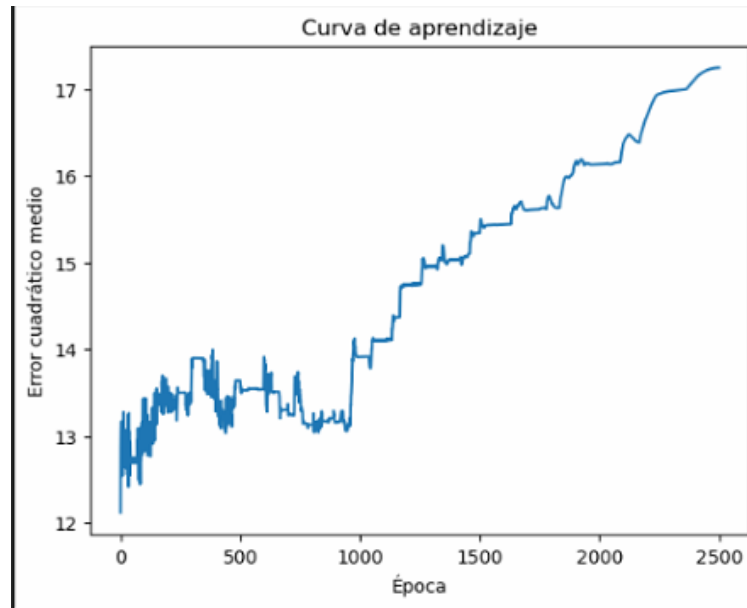


Ilustración 48 Variación del Learning Rate con Lado de mapa 20, iteraciones 5000 y eta 0.0001

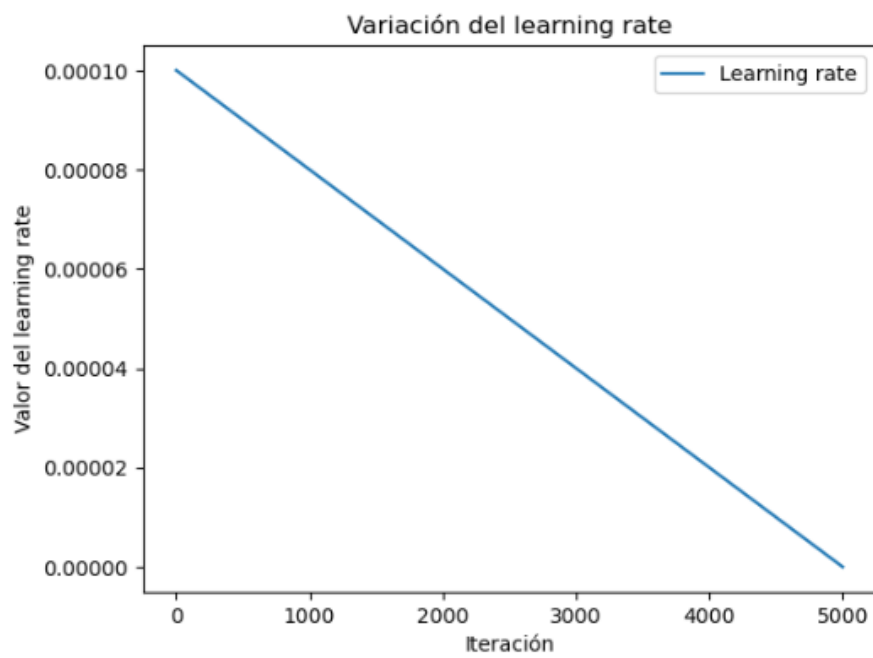


Ilustración 49 Curva de aprendizaje con Lado de mapa 20, 2500 iteraciones y 0.1 eta

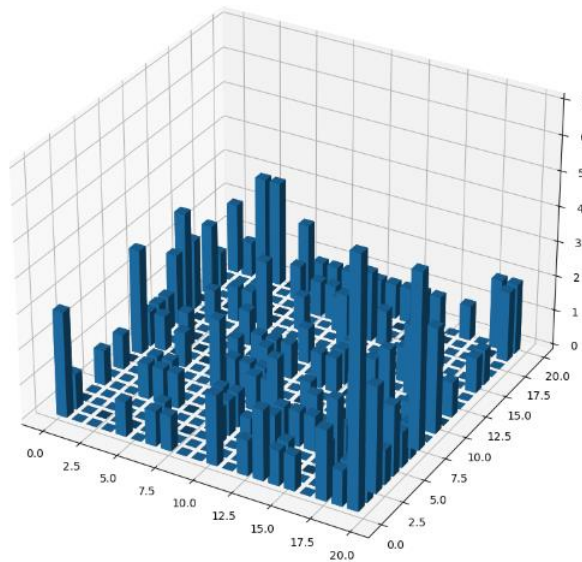




## 4.4.2 Cuestión 2

***“Para la mejor clasificación que hayas obtenido del dataset de entrenamiento, adjunta el número de clases, mapa de clasificación (incluyendo en cada neurona la lista de países), mapa de activaciones (histograma 3D), mapa de distancias del dataset, el error de cuantificación y el error topológico. Explica detalladamente los resultados. ”***

- Numero de clases existentes: los datos se agrupan en 2 clases, que son países desarrollados y subdesarrollados.
- Mapa de clasificación:
  - Observar resultado en el notebook
  - Se presenta la posición de la neurona junto a los países que ha analizado.
- Mapa de activaciones:



*Ilustración 50 Histograma del mapa de Activaciones de la Práctica 2*

Aquí se indica las neuronas que se han activado y la cantidad de veces que se ha activado cada una, viendo que como máximo se activa 7 veces una neurona, este gráfico representa como en lugar de activarse numerosas neuronas pocas veces se activan las mismas neuronas múltiples veces, propio del aprendizaje competitivo aplicado.

- Mapa de distancias:
  - Debido al tamaño de la matriz, se debe de visualizar en el cuaderno entregado.
- Error de cuantificación:
  - El error de cuantificación es una medida del grado de aproximación de una unidad (neurona) del mapa autoorganizado (SOM) a su correspondiente vector de entrada en el espacio de características. En este caso podemos observar que es un numero bajo por lo que es una aproximación cercana al patrón.  
Como se muestra en el Notebook, el error de cuantificación es: 13.13077.
- Error topológico:



- El error topológico se refiere a la discrepancia entre la topología del mapa y la topología de los datos, en este caso podemos observar que al dar el valor 0, no existe discrepancia.

#### 4.3.3 Cuestión 3

***“Observa el resultado de tu mapa de clasificación y dónde han sido clasificados los países y contesta:***

***3.1. A grandes rasgos ¿Que grupos se pueden establecer? Un grupo puede considerarse formado por varias neuronas adyacentes con países que compartan características.***

Para analizar los grupos que se forman en las distintas neuronas podemos analizar el mapa de clasificación de cada neurona con el país que las activa.

Por lo tanto, si lo observamos podemos afirmar que las neuronas que están más próximas son los países que se consideran como subdesarrollados, en las primeras neuronas, aparecen países como Haití, Gabón, Botsuana, Sudáfrica. En cambio, las más alejadas, son países más desarrollados como Italia, España, Bosnia y Herzegovina, etc.

Con todo esto, podemos afirmar que se establecen dos grupos, uno de países subdesarrollados y otro de países desarrollados.

***3.2. Analiza los vectores de pesos de las neuronas que se han activado en tu mapa para las entradas de Venezuela, Noruega y Etiopía. ¿En qué se parecen, en que se diferencian?***

Los vectores de pesos de los países representan diferentes atributos y características de estos. Cada valor indica la importancia de estos valores de cada país.

Podemos afirmar que, en Etiopía, existe un gran número de fertilidad adolescente, en cambio, el GDP per cápita muestra valores muy bajos, al igual que el GNI. Con estos dos últimos datos se puede explicar la incidencia de tuberculosis en el país, que presenta un número relativamente alto. Al tener una baja renta per cápita y un GNI bajo, la salud de los habitantes se ve mermada y es más complicado mantener las epidemias estables. El siguiente dato, porcentaje de personas que tienen acceso a agua, está altamente relacionado con el anterior, pues son números muy similares y el no tener acceso a agua puede conllevar a desarrollar más enfermedades graves. Al igual que el índice de tuberculosis, las muertes de neonatos están relacionadas con los datos ya analizados, pues si los recursos están mermados es más complicado que los neonatos sobrevivan. El siguiente dato, muertes antes de los 5 años, está ligado con todo lo comentado anteriormente, este es un número bastante alto y podemos afirmar que los neonatos que consiguen sobrevivir tienen un alto porcentaje de muerte. Además, el porcentaje de población joven de Etiopía es bastante alto y podemos afirmar que estamos ante una población muy joven. Existe una gran población rural que representa la amplia mayoría del país en cambio hay una minoría de personas que viven en zonas urbanitas. En lo que atañe a la población con acceso a internet y móviles representan una minoría. Por último, afirmamos que el GNI per cápita comparado con el internacional es muy bajo y la inflación es baja.

En lo que atañe a Venezuela, existe un bajo número de fertilidad adolescente, el GDP per cápita muestra valores muy bajos, al igual que el GNI. El porcentaje de personas que tienen acceso a agua está altamente relacionado con el índice de tuberculosis, pues esta enfermedad se debe a la contaminación del agua. Las muertes de neonatos son mucho más bajas que en Etiopía, al



igual que la mortalidad. En Venezuela existe una población con edad media, ni envejecida ni joven. Podemos afirmar que existe un gran porcentaje de población urbanita y uno pequeño de población rural. El acceso a internet y el acceso a teléfonos móviles representa un porcentaje medio-bajo de la población del país. Por último, la inflación y el GNI respecto al internacional son muy bajos.

Con los datos acontecidos en Noruega podemos afirmar que, la fertilidad adolescente es mucho más baja que en los países analizados anteriormente. El GDP es bastante alto, en comparación a los anteriores, y el GNI. Debido al acceso al agua que es muy alto, la tuberculosis no representa un dato alto en este país. La población en Noruega en su mayoría es envejecida. La mortalidad infantil es muy baja, y la población es mayor en las zonas urbanitas que en las zonas rurales. El acceso a internet y el uso de teléfonos móviles es muy alto. El más alto de los tres países. Por último, el GNI respecto al internacional es mucho más alto que en los países anteriores y observamos una inflación mucho más baja.

**3.3. Colorea el mapa de clasificación. Para ello usa la matriz de pesos y colorea cada neurona en función del valor del peso correspondiente a las características GDP per capita y Mortality rate, neonatal. (un mapa por característica). Compara ambos mapas, establece las diferencias y explícalas.**

Tras colorear el mapa de clasificación para ambas características haciendo uso de una escala de colores de azul a rojo para medir su frecuencia podemos concluir las siguientes diferencias:

- Ambas graficas son inversas, es decir, los países que tienen un GDP por cápita más bajo con aquellos que tienen una tasa de mortalidad en neonatos más alta.
- Por otro lado, aquellos países con un mayor GDP por cápita poseen una tasa de mortalidad de neonatos baja o muy baja.
- La tasa de mortalidad en neonatos se encuentra más distribuida a lo largo del gráfico, teniendo valores más dispersos mientras que los valores de GDP por cápita están más concentrados indicando.

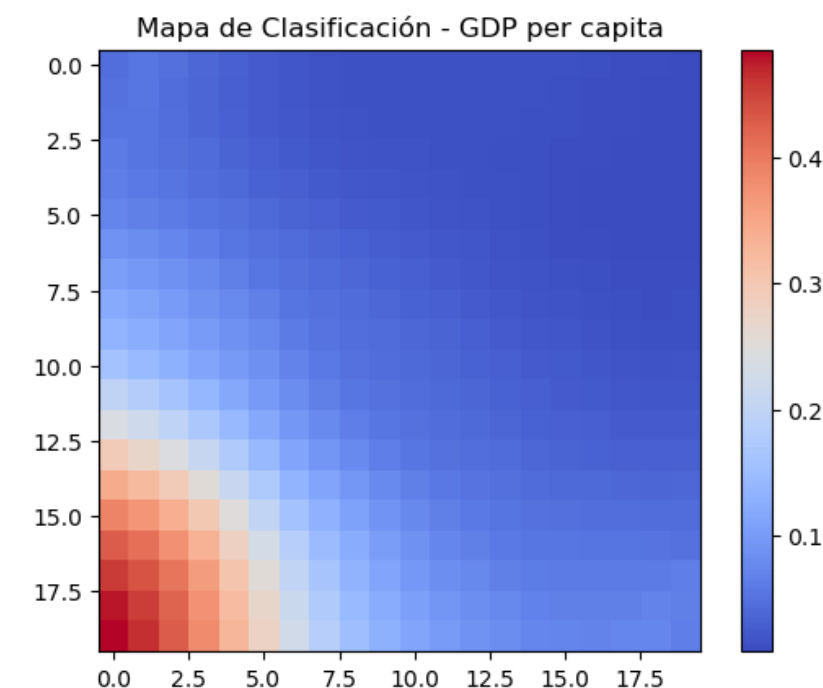


Ilustración 51 Mapa de Clasificación según GPD

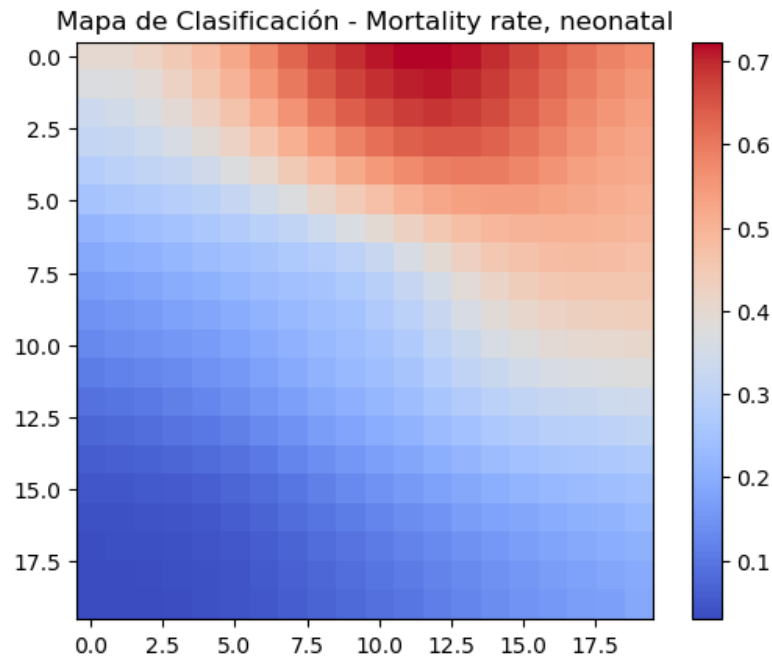


Ilustración 52 Mapa de clasificación según Mortality rate, neonatal

**3.4. ¿Cómo/dónde se encuentran los países normalmente considerados como subdesarrollados o en vías de desarrollo en tu mapa? ¿Muy concentrados o subdivididos en grupos? ¿Por qué?”**

Tal como se ha mencionado y mostrado en los gráficos del apartado anterior los países considerados subdesarrollados se encuentran en la zona de un GDP per cápita bajo o muy bajo y una ratio de mortalidad neonatal alto, además de estar situados en esta zona se puede observar cómo los países respecto al GDP per cápita se encuentran concentrados pues comparte la característica de un GDP per cápita bajo, pero son más dispersos en la característica de la ratio de mortalidad.

Esto se debe a que el GDP per cápita es inversamente proporcional a la mortalidad.

**4.4.4 Cuestión 4 Optativa**

**“Usa el mapa de clasificación coloreado para GDP per cápita para colorear un mapamundi donde cada país tendrá el color de la neurona que se ha activado al clasificarlo”**

Lo primero que realizamos es guardar en diferentes archivos, la matriz de pesos total, los vectores de pesos de la columna correspondiente al gdp, los nombres de los países y los datos de gdp del archivo csv. Esto lo hacemos para poder usar estos datos de una forma más profesional de cara a crear el mapa.



## Mapa

```
if(os.path.exists("mapa")) == 0:
    os.mkdir("mapa")
np.save('mapa/gdp_weights.npy', gdp_weights)
np.save('mapa/pesos.npy', matriz_pesos)
np.save('mapa/nombresPaíses.npy', np.array(datos.CountryName))
np.save('mapa/gdp_data.npy', np.array(datos.GDP_per_capita_current_US))
```

Ilustración 53 Guardar datos del mapa

Lo segundo que se realiza es cargar los archivos previamente creados en arrays. A continuación, descargamos el dataset de geopandas que contiene países y sus correspondientes coordenadas para poder localizarlos en el mapamundi.

```
# cargar el modelo de red neuronal previamente entrenado
model = np.load('mapa/pesos.npy')

# cargar los datos de GDP per capita y nombres para cada país
gdp_pesos = np.load('mapa/gdp_weights.npy', allow_pickle = True)
countries = np.load('mapa/nombresPaíses.npy', allow_pickle = True)
gdp_data = np.load('mapa/gdp_data.npy', allow_pickle = True)

# cargamos el dataset de geopandas
world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
datosMapa = {'name': [], 'Neurona': []}
# creamos las columnas que nos van a hacer falta
world["Neurona"] = np.zeros(len(world['name']))
world["ColorR"] = np.zeros(len(world['name']))
world["ColorG"] = np.zeros(len(world['name']))
world["ColorB"] = np.zeros(len(world['name']))

# Usamos el diccionario que guarda los países que reconoce cada neurona para generar un mismo colorRGB
# aleatorio en los países de cierta neurona
for clave, array in mapa_clasificacion_Modificado.items():
    for i in array:
        rand1 = np.random.random(size=1)
        rand2 = np.random.random(size=1)
        rand3 = np.random.random(size=1)
        for j in range(len(world['name'])):
            if world['name'][j] == i:
                world['Neurona'][j] = clave
                world['ColorR'][j] = rand1
                world['ColorG'][j] = rand2
                world['ColorB'][j] = rand3
```

Ilustración 54 Creación y llenado del diccionario de datos de world

Usando el diccionario de mapa clasificación, el cual contiene los países que reconoce cada neurona, creamos una columna para saber la posición de la neurona y otras tres columnas para cada canal del color. Los países que sean detectados por la misma neurona tendrán el mismo color.



```
# Plotear el mapa del mundo con colores según GDP per capita
fig = plt.figure(figsize=(16, 8))
m = Basemap(projection='mill', llcrnrlat=-60, urcrnrlat=90, llcrnrlon=-180, urcrnrlon=180)
m.drawcoastlines()
m.drawmapboundary()
m.fillcontinents(color='lightgray', lake_color='lightgray')
m.drawparallels(range(-90, 90, 30), labels=[1, 0, 0, 0])
m.drawmeridians(range(-180, 180, 60), labels=[0, 0, 0, 1])

#Recorre el dataset para pintar cada punto del país con su color correspondiente
for idx, row in world.iterrows():
    x, y = m(row.geometry.centroid.x, row.geometry.centroid.y)

    colorR = row['ColorR']
    colorG = row['ColorG']
    colorB = row['ColorB']
    num_color = [colorR, colorG, colorB]
    m.plot(x, y, 'o', markersize=4, color=num_color, alpha=0.5)
plt.title('GDP per capita')
plt.show()
```

Ilustración 55 Creación y coloreo del mapamundi

Por último, creamos la figura del mapa y a cada país del dataset de geopandas le asignamos los colores asociados a dicha neurona para poder el punto del país. Esta librería no permite pintar el país completo, únicamente permite los puntos con las coordenadas del dataset.

La salida que genera el código es el siguiente mapa el cual, cada punto representa un país en el que se encuentra y el color la neurona que se ha activado. Podemos observar casos como que en Europa hay bastantes colores similares en azul ya que entrarían en la clase desarrollados. En Sudamérica hay varios puntos de color rosa por lo que podrían también pertenecer a la misma clase y se puede observar también que en África hay muchos colores de tonalidad marrón iguales que podría pertenecer a una clase también.

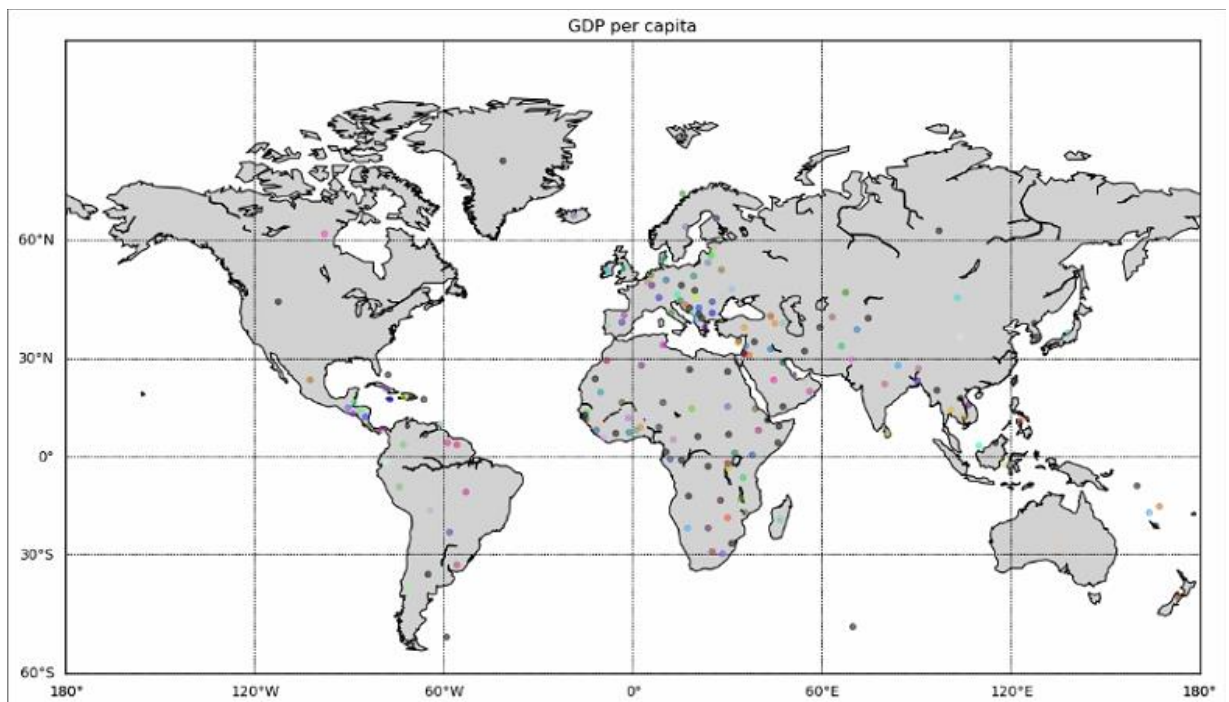


Ilustración 56 Mapamundi con sus neuronas



#### 4.5 Conclusiones

En conclusión, la creación de una red SOM para la clasificación de países según sus parámetros es una tarea desafiante pero altamente efectiva en términos de obtener una comprensión holística y visual de las características de los países.

La utilización de esta herramienta permite la identificación de patrones complejos y relaciones ocultas entre los parámetros de los países, lo que puede ser útil para tomar decisiones y estrategias en diversos campos, como el comercio, la política y la cooperación internacional. Sin embargo, es importante tener en cuenta que la precisión de la red SOM dependerá en gran medida de la calidad y cantidad de datos disponibles, así como de la selección de parámetros relevantes para el análisis.

En general, la implementación de una red SOM para la clasificación de países es una valiosa herramienta para la comprensión de la complejidad del mundo actual y para la toma de decisiones informadas.



## 5.- Anexo A

En este anexo adjuntamos las conversaciones con GPT3.


como guardarías un dato más en una matriz de activaciones?

Para guardar un dato más en una matriz de activaciones, se puede seguir los siguientes pasos:

1. Asegurarse de que la matriz tenga suficiente capacidad para almacenar el nuevo dato. Es decir, la matriz debe tener una dimensión que permita agregar una nueva fila o columna para el nuevo dato, dependiendo de cómo esté estructurada la matriz.
2. Determinar la posición en la que se va a almacenar el nuevo dato. Si se está agregando una nueva fila, se puede ubicar en la última posición de la matriz. Si se está agregando una nueva columna, se puede ubicar en la última posición de cada fila existente.
3. Asignar el valor del nuevo dato a la posición determinada en el paso anterior. Esto se puede hacer mediante la asignación directa del valor en la posición correspondiente, o mediante una función o método específico dependiendo del lenguaje de programación que se esté utilizando.

Por ejemplo, en Python se puede agregar un nuevo dato a una matriz de activaciones de la siguiente manera:

makefile

 Copy code

```
# Definir una matriz de activaciones existente
activaciones = [[1, 0, 1], [0, 1, 0], [1, 1, 1]]

# Agregar una nueva fila con los valores de [0, 0, 1]
nuevo_dato = [0, 0, 1]
activaciones.append(nuevo_dato)

# Agregar una nueva columna con los valores de [1, 0, 1]
for fila in activaciones:
    fila.append(1)
```

En este ejemplo, se agrega un nuevo dato a la matriz de activaciones. Primero se define la matriz existente, y luego se agrega una nueva fila mediante el método `append()`, que agrega el valor de `nuevo_dato` a la matriz. Luego, se agrega una nueva columna mediante un ciclo `for` que recorre cada fila de la matriz y agrega un valor de `1` al final de cada fila existente.





j haz una función en python para pintar la variación del learning rate de un som

Para graficar la variación del learning rate de un SOM en Python, se puede usar la librería `matplotlib`. A continuación, se proporciona una función que recibe como entrada la lista de valores del learning rate en cada iteración y grafica su evolución a lo largo del tiempo:

```
python Copy code

import matplotlib.pyplot as plt

def plot_learning_rate(lr_rate):
    """
    Esta función grafica la variación del learning rate en cada iteración.
    :param lr_rate: una lista de valores de learning rate en cada iteración
    """
    # Crear la figura
    fig, ax = plt.subplots()

    # Crear el gráfico de líneas
    ax.plot(lr_rate, label='learning rate')

    # Añadir etiquetas y título
    ax.set_xlabel('Iteración')
    ax.set_ylabel('Valor del learning rate')
    ax.set_title('Variación del learning rate')

    # Mostrar la leyenda y el gráfico
    ax.legend()
    plt.show()
```

j como puedo guardar una imagen en python cada vez que es generada con un plt?

Para guardar una imagen generada con matplotlib.pyplot en Python, puedes usar la función `savefig()` de la biblioteca. La sintaxis básica es la siguiente:

```
python Copy code

import matplotlib.pyplot as plt

# Genera tu gráfico
plt.plot([1, 2, 3], [4, 5, 6])

# Guarda tu gráfico
plt.savefig('nombre_del_archivo.png')
```

En este ejemplo, se genera un gráfico simple con `plt.plot()`, y luego se guarda en un archivo PNG con `plt.savefig()`. El nombre del archivo que desees guardar debe ser pasado como argumento a `savefig()`. También puedes especificar la ubicación del archivo y otros parámetros opcionales para el formato de imagen y la calidad de la imagen.

Ten en cuenta que debes llamar a `savefig()` después de generar tu gráfico y antes de llamar a cualquier otra función de pyplot que pueda afectar la figura, como `show()`.



si guardo una serie de figuras como puedo mostrarlas luego todas seguidas en forma de GIF??

Para mostrar una serie de figuras seguidas en forma de animación GIF, puedes utilizar la biblioteca `imageio`. Primero, asegúrate de haber instalado esta biblioteca en tu entorno de Python. Puedes instalarlo ejecutando el siguiente comando en tu terminal o consola de comandos:

[Copy code](#)

```
pip install imageio
```

A continuación, puedes utilizar la siguiente plantilla de código para guardar tus figuras generadas por `matplotlib` como imágenes y luego crear una animación GIF a partir de esas imágenes:

python

[Copy code](#)

```
import matplotlib.pyplot as plt
import imageio

# Genera y guarda las figuras
for i in range(10):
    fig, ax = plt.subplots()
    ax.plot([0, 1], [i, i+1])
    plt.savefig(f"imagen_{i}.png")

# Crea la animación
with imageio.get_writer('animacion.gif', mode='I') as writer:
    for i in range(10):
        filename = f"imagen_{i}.png"
        image = imageio.imread(filename)
        writer.append_data(image)
```

Este código generará 10 figuras y las guardará como archivos de imagen en el directorio actual. Luego, creará una animación GIF a partir de esas imágenes y la guardará como `animacion.gif` en el mismo directorio. Puedes ajustar el número de figuras y el nombre del archivo de imagen para adaptarlo a tus necesidades.



Puedo sacar un video directamente con esta funcion??

La función que tienes actualmente produce un archivo de animación en formato GIF, que es un formato de imagen animada. Si deseas producir un video en lugar de un GIF, debes cambiar el formato de salida que está especificado en la línea donde se crea el objeto `writer`. En lugar de `'I'`, que es el código de formato para un archivo de animación GIF, puedes especificar un formato de video compatible con `imageio`.

Por ejemplo, si quieres guardar un video en formato MP4, puedes cambiar la línea:

python

Copy code

```
with imageio.get_writer('animacion.gif', mode='I') as writer:
```

a:

python

Copy code



```
with imageio.get_writer('animacion.mp4', fps=30) as writer:
```

Aquí, el argumento `fps` establece la velocidad de cuadros por segundo del video. En este caso, se establece en 30 fps.


Una vez que haya hecho este cambio, la función debería generar un archivo de video en lugar de un archivo de animación GIF.



como puedo crear un mapamundi en python para luego colorearlo?

Para crear un mapamundi en Python y colorearlo, puedes utilizar la biblioteca ``Basemap`` de `matplotlib`. Esta biblioteca es útil para crear mapas geográficos y geoespaciales y puede ser utilizada para visualizar datos geográficos.  

A continuación, te muestro un ejemplo básico de cómo crear un mapa del mundo y colorear algunos países utilizando la biblioteca ``Basemap``:

```
python  Copy code

from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt

# Crear la figura y los ejes del mapa
fig = plt.figure(figsize=(8, 8))
m = Basemap(projection='ortho', resolution=None, lat_0=50, lon_0=-100)

# Dibujar los países y las costas
m.shadedrelief()
m.drawcoastlines(color='gray')
m.drawcountries(color='gray')

# Colorear algunos países
m.fillcontinents(color='green', lake_color='blue')
m.drawmapboundary(fill_color='blue')

# Mostrar el mapa
plt.show()
```

En este ejemplo, se crea una figura y los ejes del mapa utilizando la proyección ortográfica (``projection='ortho'``) centrada en las coordenadas latitud-longitud (50, -100). Luego, se dibujan los países y las costas utilizando ``m.drawcoastlines()`` y ``m.drawcountries()``.

Para colorear algunos países, se utiliza ``m.fillcontinents()`` con el color verde para los continentes y el color azul para los lagos. Finalmente, se muestra el mapa utilizando ``plt.show()``.

Este es solo un ejemplo básico para crear un mapa del mundo en Python utilizando la biblioteca ``Basemap``. Puedes personalizar el mapa y agregar más detalles según tus necesidades.



haz una funcion para imprimir la curva de aprendizaje de un som

Claro, aquí te dejo una posible implementación de la función para imprimir la curva de aprendizaje de un SOM:



python

Copy code

```
import matplotlib.pyplot as plt
import numpy as np

def plot_curva_aprendizaje(mse):
    """
    Función que grafica la curva de aprendizaje de un SOM.

    Args:
    mse: ndarray, array que contiene el error cuadrático medio por época.
    """
    plt.plot(range(len(mse)), mse)
    plt.title('Curva de aprendizaje')
    plt.xlabel('Época')
    plt.ylabel('Error cuadrático medio')
    plt.show()
```

Para usarla, simplemente tendrías que pasarle como argumento el array con los valores de error cuadrático medio por época:



python

Copy code

```
# Entrenar SOM y calcular curva de aprendizaje
mse = entrenar_som(datos, matriz_pesos, periodo)
# Imprimir curva de aprendizaje
plot_curva_aprendizaje(mse)
```




como funciona la funcion map de la biblioteca geopandas

La función `map` de la biblioteca `geopandas` se utiliza para transformar una columna de un `GeoDataFrame` (una estructura de datos similar a un `DataFrame` de `Pandas`, pero que incluye información espacial) aplicando una función o un diccionario a cada valor de la columna.  

La sintaxis básica de la función es la siguiente:

scss


 Copy code

```
geodf['columna_nueva'] = geodf['columna_original'].map(funcion_o_diccionario)
```

Donde `geodf` es el `GeoDataFrame`, `columna_nueva` es el nombre de la nueva columna que se quiere crear, `columna_original` es el nombre de la columna existente que se quiere transformar y `funcion_o_diccionario` es la función o el diccionario que se aplicará a cada valor de la columna.

Si se utiliza una función, esta debe tomar como entrada cada valor de la columna original y devolver un valor transformado. Por ejemplo, si se tiene una columna de altitudes en metros y se quiere convertirla a pies, se podría utilizar la siguiente función:


python

 Copy code

```
def metros_a_pies(metros):  
    pies = metros * 3.28084  
    return pies
```


Y luego se puede aplicar la función a la columna de altitudes de la siguiente manera:

scss

 Copy code

```
geodf['altitud_pies'] = geodf['altitud_metros'].map(metros_a_pies)
```

Por otro lado, si se utiliza un diccionario, este debe tener como claves los valores originales de la columna y como valores los valores transformados. Por ejemplo, si se tiene una columna de tipos de suelos que utiliza códigos numéricos y se quiere crear una nueva columna con los nombres de los tipos de suelos, se podría utilizar el siguiente diccionario:

 Regenerate response



como hago una curva de aprendizaje en python para un som?

Para hacer una curva de aprendizaje en Python para una red SOM, se puede utilizar la librería `MiniSom`. A continuación, se proporciona un ejemplo de cómo se puede construir y graficar la curva de aprendizaje de una red SOM en Python:



python

Copy code

```
from minisom import MiniSom
import numpy as np
import matplotlib.pyplot as plt

def plot_learning_curve(som, X, y, num_iterations):
    """
    Esta función entrena una red SOM y grafica la curva de aprendizaje.
    :param som: instancia de la clase MiniSom
    :param X: matriz de características de entrada
    :param y: vector de etiquetas de salida
    :param num_iterations: número de iteraciones de entrenamiento
    """

    som.pca_weights_init(X)
    som.train_random(X, num_iterations, verbose=False)



    # Calcular el error cuadrático medio de cada iteración
    mse = []
    for i in range(num_iterations):
        e = np.mean(np.linalg.norm(X - som.win_map(X)[i], axis=1))
        mse.append(e)

    # Graficar la curva de aprendizaje
    plt.plot(range(num_iterations), mse)
    plt.title('Curva de aprendizaje')
    plt.xlabel('Iteración')
    plt.ylabel('Error cuadrático medio')
    plt.show()


# Ejemplo de uso de la función plot_learning_curve
data = np.genfromtxt('iris.csv', delimiter=',', usecols=(0, 1, 2, 3))
labels = np.genfromtxt('iris.csv', delimiter=',', usecols=(4), dtype=str)
som = MiniSom(7, 7, 4, sigma=1.0, learning_rate=0.5)
plot_learning_curve(som, data, labels, 1000)
```



como puedo guardar una imagen en python cada vez que es generada con un plt?

Para guardar una imagen generada con matplotlib.pyplot en Python, puedes usar la función `plt.savefig()` de la biblioteca. La sintaxis básica es la siguiente:  

python

 Copy code

```
import matplotlib.pyplot as plt

# Genera tu gráfico
plt.plot([1, 2, 3], [4, 5, 6])

# Guarda tu gráfico
plt.savefig('nombre_del_archivo.png')
```

En este ejemplo, se genera un gráfico simple con `plt.plot()`, y luego se guarda en un archivo PNG con `plt.savefig()`. El nombre del archivo que desees guardar debe ser pasado como argumento a `savefig()`. También puedes especificar la ubicación del archivo y otros parámetros opcionales para el formato de imagen y la calidad de la imagen.

Ten en cuenta que debes llamar a `savefig()` después de generar tu gráfico y antes de llamar a cualquier otra función de pyplot que pueda afectar la figura, como `show()`.





si guardo una serie de figuras como puedo mostrarlas luego todas seguidas en forma de GIF??

Para mostrar una serie de figuras seguidas en forma de animación GIF, puedes utilizar la biblioteca ``imageio``. Primero, asegúrate de haber instalado esta biblioteca en tu entorno de Python. Puedes instalarlo ejecutando el siguiente comando en tu terminal o consola de comandos:

[Copy code](#)

```
pip install imageio
```

A continuación, puedes utilizar la siguiente plantilla de código para guardar tus figuras generadas por ``matplotlib`` como imágenes y luego crear una animación GIF a partir de esas imágenes:

python

[Copy code](#)

```
import matplotlib.pyplot as plt
import imageio

# Genera y guarda las figuras
for i in range(10):
    fig, ax = plt.subplots()
    ax.plot([0, 1], [i, i+1])
    plt.savefig(f"imagen_{i}.png")

# Crea la animación
with imageio.get_writer('animacion.gif', mode='I') as writer:
    for i in range(10):
        filename = f"imagen_{i}.png"
        image = imageio.imread(filename)
        writer.append_data(image)
```

Este código generará 10 figuras y las guardará como archivos de imagen en el directorio actual. Luego, creará una animación GIF a partir de esas imágenes y la guardará como ``animacion.gif`` en el mismo directorio. Puedes ajustar el número de figuras y el nombre del archivo de imagen para adaptarlo a tus necesidades.



## 6.- Bibliografía

*Introduction to Self-Organizing Maps" de Anders Boesen Lindbo Larsen*  
(artículo). (s/f).

Python, R. (2021, marzo 17). *Python AI: How to build a neural network & make predictions*. Realpython.com; Real Python.

<https://realpython.com/python-ai-neural-network/>

(N.d.). Openai.com. Retrieved February 19, 2023, from  
<https://chat.openai.com/chat>