Redes Neuronales

TP1

Entrene una red de Hopfield ‘82 con las imágenes binarias disponibles en el campus.

1. Verifique si la red aprendió las imágenes enseñadas.
2. Evalúe la evolución de la red al presentarle versiones alteradas de las imágenes aprendidas: agregado de ruido, elementos borrados o agregados.
3. Evalúe la existencia de estados espurios en la red: patrones inversos y combinaciones de un número impar de patrones. (Ver Spurious States, en la sección 2.2, Hertz, Krogh & Palmer, pág. 24).
4. Realice un entrenamiento con las 6 imágenes disponibles. ¿Es capaz la red de aprender todas las imágenes? Explique.

Lo primero que voy a hacer es importar las bibliotecas básicas para poder desarrollar el ejercicio.

In [8]:
```
# primero importamos numpy y algo para leer imágenes y hacer graficos
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
```

Para las imágenes opté la manera fácil de usarlas locales, pero no sería difícil descargarlas desde gitHub en runtime y usarlas de ahí. Para los incisos 1, 2 y 3 separé las imágenes en 2 grupos porque son de diferentes tamaños. El cuarto se hace con las 6 imágenes con alguna modificación a los tamaños.

In [9]:
```
# necesito paths a las imagenes,vienen de un repo clonado localmente
path1 = 'imagenes_tp1\\paloma.bmp'
path2 = 'imagenes_tp1\\panda.bmp' # 50*50
path3 = 'imagenes_tp1\\perro.bmp' #50*50
path4= 'imagenes_tp1\\quijote.bmp'
path5= 'imagenes_tp1\\torero.bmp'
path6= 'imagenes_tp1\\v.bmp' #50*50
```

Creo una función que toma una lista de paths a las imágenes y devuelve las imágenes en forma de vectores columna concatenadas.

In [10]:
```
def imgPath_to_vector(paths):
    """La idea es tomar una lista de paths y generar una matriz cuyas columnas sean
    list = []
    for path in paths:
        img = Image.open(path)
        vector_img = np.array(img).astype(np.int8)
        forma = np.shape(vector_img)
        vector_img = vector_img.ravel()#ideal del profe, tratar las imagenes como v
        list.append(vector_img *2 -1) # acá lo ajusto para ir de (0,1)->(-1,1)
    return np.asarray(list).transpose(), list,forma

flattened_images1, lista_img1,forma1 = imgPath_to_vector([path2,path3,path6]) # ten
flattened_images2, lista_img,forma2 = imgPath_to_vector([path1,path4,path5])
```

La altura de estos vectores columna (nuestras "realizaciones" para el contexto de lo que buscamos que aprendan las neuonas) nos da la cantidad de neuronas y, por ende, el tamaño de la matriz W de pesos de interconexión de neuronas. Como se discutió en clase, la diagonal de W son ceros para que las nueronas no se realimenten a si mismas.

Para este primer caso, el factor de aprendizaje η se mantiene unitario, no afecta al aprendizaje.

La matriz W se compone de diferentes $w_{i,j}$, inicializados en cero (van de -1 a 1) y actualizados según la regla:

$$\Delta w_{i,j_n} = \eta \cdot p_i^1 \cdot p_j^1$$

$$w_{i,j_{n+1}} = w_{i,j_n} + \Delta w_{i,j_n}$$

Dónde p_i^1 es el pattern\patrón en la iteración 1 y posición i.

La matriz se calcula por medio de un producto matricial entre los vectores columna de datos con ellos mismos.

In [11]:
```
# ahora sabemos que hay tantas neuronas como valores en los vectores columna que co
# W es una matriz cuadrada de filas y columnas de igual tamaño a la cantidad de neu

def calcular_matriz_W(flattened_images,eta=1):
    filas = np.shape(flattened_images)[0] # cant de neuronas
    columnas = np.shape(flattened_images)[1] # cant de imagenes (vectores)
    W_matrix = np.zeros([filas,filas]) # creo la matriz W,de pesos

    #for i in range(columnas): # itera 1 vez por cada pattern
    #    delta_W = eta * np.asmatrix(flattened_images[:,i]).T @ np.asmatrix(flatten
    #    W_matrix = W_matrix + delta_W

    delta_W = eta * np.asmatrix(flattened_images) @ np.asmatrix(flattened_images).T
    W_matrix = W_matrix + delta_W

    W_matrix = W_matrix - np.diag(np.diag(W_matrix)) # le saco la diagona,lo de dia
    return W_matrix
```

Ahora calculo la matriz de pesos para el primer set de imágenes/patrones. Con este, podemos ver la evolución de la red neuronal partiendo del patrón que se quiera. Para el primer inciso se le alimentan los mismos patrones y se revisa que no cambie entre iteraciones singulares, indicando que son mínimos en la función que representa los estados posibles de la red. En otras palabras, si no cambia es porque se aprendió el patrón.

In [12]:
```
W_matrix = calcular_matriz_W(flattened_images1)
```

In [13]:
```
def correr_red(W, patron, iters=5):
    """Para dejar que la red vaya a algún estado desde el patron"""
    estado = patron.copy() # copia el patron como estado inicial de la red
    for _ in range(iters): # itera n veces
        estado = np.sign(W @ estado) # porahora calcula todas juntas, aunque eso pu
    return estado
```

In [14]:
```
def chequear_aprendizaje(W_matrix,flattened_images,iters=5):
    """Esta función solo recibe patrones y devuelve a donde se fue la red "iters" i
    columnas = np.shape(flattened_images)[1] # cant de imagenes (vectores)
    lista_recordada = [] #para guardar lo que recuerda

    for i in range(columnas): # itera 1 vez por cada imagen que se aprendió
        original = np.asmatrix(flattened_images[:, i]).T
        res = correr_red(W_matrix,original , iters)
        lista_recordada.append(res)

        if np.array_equal(original, res):# avisamos si las imágenes se recordaron b
            print(f"La imagen {i} fue recordada correctamente.")
        else:
            print(f"La imagen {i} NO fue recordada correctamente.")

    return lista_recordada
```

In [15]:
```python
# ahora tocaría poner la imagen real al lado de la memoria recuperada
def mostrar_resultados(memoria, imagenes,forma):
    """Función que muestra las imágenes reales arriab y a las que converge la red a

    En memoria van las resultantes de iterar, en imágenes las reales, en forma va e
    """
    alto, ancho = forma[0], forma[1]
    n_imagenes = len(memoria)

    for i in range(n_imagenes):
        plt.subplot(2, n_imagenes, i + 1)
        plt.imshow(imagenes[:, i].reshape(alto, ancho), cmap='gray')
        plt.axis('off')
        if i == 0:  # Agregar título solo en la primera imagen
            plt.title('Imágenes originales')

        plt.subplot(2, n_imagenes, i + 1 + n_imagenes)
        plt.imshow(memoria[i].reshape(alto, ancho), cmap='gray')
        plt.axis('off')
        if i == 0:  # Agregar título solo en la primera imagen
            plt.title('Imágenes después de iterar')

    plt.tight_layout()
    plt.show()
```

Resultados del inciso 1

Con todas las funciones ya definidas revisamos que las redes hayan aprendido.

In [16]:
```
memoria1 = chequear_aprendizaje(W_matrix, flattened_images1)
```

```
La imagen 0 fue recordada correctamente.
La imagen 1 fue recordada correctamente.
La imagen 2 fue recordada correctamente.
```

In [17]:
```
# idem con el 2do set de imagenes
W_matrix2 = calcular_matriz_W(flattened_images2)
memoria2 = chequear_aprendizaje(W_matrix2, flattened_images2)
```

```
La imagen 0 fue recordada correctamente.
La imagen 1 fue recordada correctamente.
La imagen 2 fue recordada correctamente.
```

Se observa que ambas redes entrenadas aprendieron los patrones/imágenes. Abajo se muestran comparaciones de laas imágenes reales y las aprendidas. En este caso son idénticas, pero no lo va a ser para el segundo inciso.

In [18]:
```
mostrar_resultados(memoria1, flattened_images1,forma1)
```

In [19]:
```
mostrar_resultados(memoria2, flattened_images2,forma2)
```

Imágenes originales

Imágenes después de iterar