Trabajo Práctico Nº1

[86.54] Redes Neuronales
2ºC 2024

Alumno	Padrón	Correo
Julián Stejman	102840	jstejman@fi.uba.ar

Consignas

1. Entrene una red de Hopfield ‘82 con las imágenes binarias disponibles en el campus.
   a) Verifique si la red aprendió las imágenes enseñadas.
   b) Evalúe la evolución de la red al presentarle versiones alteradas de las imágenes aprendidas: agregado de ruido, elementos borrados o agregados.
   c) 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).
   d) Realice un entrenamiento con las 6 imágenes disponibles. ¿Es capaz la red de aprender todas las imágenes? Explique.
2. a) Comprobar estadísticamente la capacidad de la red Hopfield ’82 calculando la cantidad máxima de patrones pseudo-aleatorios aprendidos en función del tamaño de la red. Obtener experimentalmente los resultados de la siguiente tabla (los valores de la tabla corresponden a una iteración con actualización sincrónica).

P_{error}	P_{max}/N
0.001	0.105
0.0036	0.138
0.01	0.185
0.05	0.37
0.1	0.61

   b) Proponga una manera de generar patrones con distintos grados de correlación. Utilice el método propuesto para analizar cómo varía la capacidad de la red Hopfield en función de la correlación entre patrones.

Desarrollo

0.1. Ejercicio 1

Inicialmente, para poder entrenar y evaluar a una red de Hopfield necesito una manera de convertir las imágenes en blanco y negro en patrones que pudieran ser aprendidas por la red.

Por ese motivo se desarrollan 3 funciones auxiliares necesarias para este ejercicio. Una función se encarga de asegurar que todas las imágenes tengan las mismas dimensiones agrandando a todas las imágenes con bandas negras. En este caso se redimensionan todas las imágenes hacia 60x50 pixeles lo cual la red destinará 3000 neuronas para aprender.

Las otras dos funciones se encargan de convertir las imágenes en vectores de 1’s y -1s (negro y blanco) y recibir un vector de 1’s y -1s y convertirlo en una imagen.

```
import numpy as np
from PIL import Image, ImageOps
import matplotlib.pyplot as plt

def pad_image(file_path, target_size=(60, 50)):
    image = Image.open(file_path)
    width, height = image.size
    pad_width = (target_size[0] - width) // 2
    pad_height = (target_size[1] - height) // 2
    padded_image = ImageOps.expand(image, border=(pad_width, pad_height), fill=0)
    padded_image = padded_image.crop((0, 0, target_size[0], target_size[1]))
    return padded_image

def bmp_to_list(image):
    array = np.array(image)
    array = np.where(array == 0, -1, array)
    array = array.ravel()
    array = array.tolist()
    for i in range(len(array)):
        if array[i] == -1:
            array[i] = 1
        else:
            array[i] = -1

    return array

def create_composite_image(image, target_size = (60, 50)):

    image_array = np.array(image)
    image_array = 255 - image_array
    reshaped_image = image_array.reshape(target_size[1], target_size[0])

    plt.figure(figsize=(5, 5))
    plt.imshow(reshaped_image, cmap='gray')
    plt.show()
```

Se desarrolla la definición de clase para la red de Hopfield con los métodos que harían que la red aprendiera y que la red predijera la salida.

```
    class HopfieldNetwork:
    def __init__(self, num_neurons):
        self.num_neurons = num_neurons
        self.weights = np.zeros((num_neurons, num_neurons))
    def train(self,patterns, eta = 1):
        for pattern in patterns:
            self.weights += eta * np.outer(pattern, pattern)
        np.fill_diagonal(self.weights, 0)
    def run(self, pattern):
        new_pattern = pattern.copy()
        indexes = np.random.permutation(self.num_neurons)
        for index in indexes:
```

```
            weighted_value = np.dot(self.weights[index], new_pattern)
            new_pattern[index] = 1 if weighted_value > 0 else -1
        return new_pattern
```

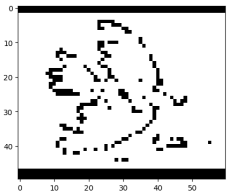
Con esto ya podemos evaluar si la red aprendió correctamente. Se la entrena:

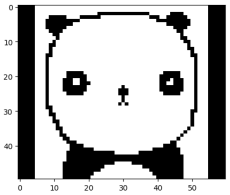
```
num_neurons = 60 * 50
num_patterns = len(image_list)

hopfield_network = HopfieldNetwork(num_neurons)
hopfield_network.train(image_list, 200)
```

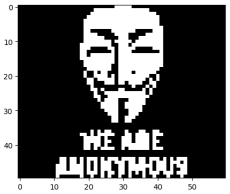
Luego del entrenamiento se le introduce las imágenes originales para ver como las reproduce la red:

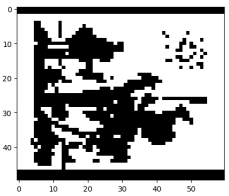
```
    for image in image_list:
        create_composite_image(hopfield_network.run(image))
```

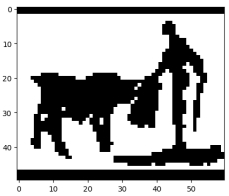
Aquí se ve la salida de la red interpretada convertida a imagen.











Para estas imágenes se puede ver que pudo aprenderlas y reproducirlas aunque para la paloma por algún motivo se le borró parte de su definición.

Ahora a cada imagen se le agregará una banda horizontal de 200 px donde en esa banda los colores de la imagen se invierten.

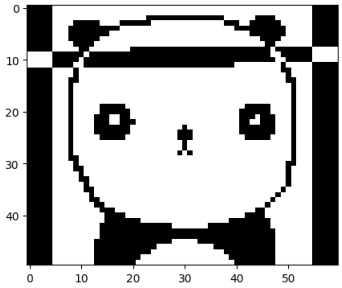


Figura 1: Ejemplo de imagen con 200px de inversión

Luego de correr esta imagen por la red se obtiene:

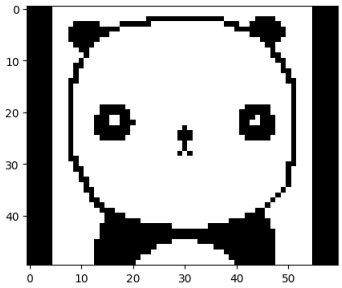


Figura 2: Imagen reconstruida por red Hopfield

Todas las imágenes con esta inversión se pueden recrear exactamente igual que si se entregase la imagen inicial como patrón en la red. En cambio, si se agranda esa banda de inversión se empiezan a generar imágenes que parecen mezclas de imágenes, cayendo en estados espurios y sin poder identificar correctamente.

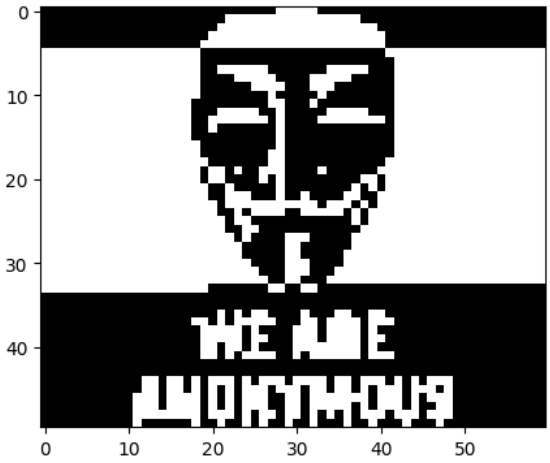


Figura 3: Ejemplo de imagen con banda de inversión de 1700 pixeles

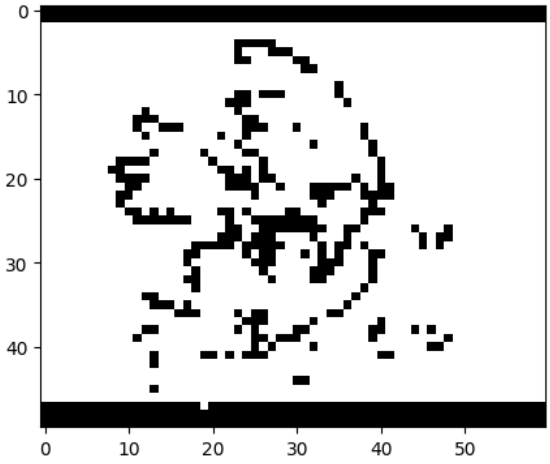


Figura 4: Reconstrucción incompleta de red Hopfield

Se puede ver que la salida de este patrón es una mezcla entre la paloma y la imagen de Anonymous. Dada la falta de información y ambigüedad de la imagen enviada a la red, la recomposición de imagen queda a mitad de camino y sin poder identificar correctamente que imagen fue. Esto da a entender que, al menos para estas redes, se necesita una cantidad mínima de información para correctamente reconstruir y, en sí, clasificar a la imagen.

0.2. Ejercicio 2

Para este ejercicio se define un método para correr en base a un patrón de manera sincrónica, o sea, que las neuronas se entrenen todas al mismo tiempo y en orden.

```
def run_syncronically(self, pattern):
    new_pattern = pattern.copy()
    weighted_value = np.dot(self.weights, new_pattern)
    new_pattern = np.where(weighted_value > 0, 1, -1)
    return new_pattern
```

Luego se construye una red de 300 neuronas y 500 patrones donde los valores son elegidos al azar entre 1 y -1. Se entrena a la red de a un patrón por vez y se almacenan los errores totales respecto a la cantidad de neuronas. Lo que se busca es ver cuántos patrones se pueden aprender, mientras se hiciera actualización sincrónica, antes de llegar al tope de error propuesto. De esta manera, se puede concluir cuál es la capacidad de patrones para determinada cantidad de neuronas.

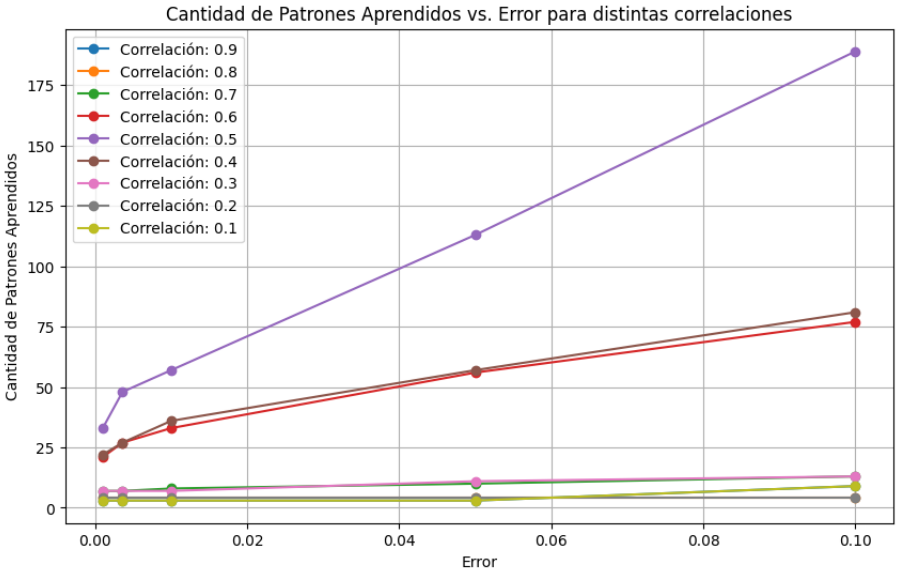
Error	# Patrones	Neuronas	#Patrones/N	#Patrones/N (teórico)	Diferencia
0.001	32	300	0.1067	0.105	0.0017
0.0036	44	300	0.1467	0.138	0.0087
0.01	58	300	0.1933	0.185	0.0083
0.05	109	300	0.3633	0.37	0.0067
0.1	184	300	0.6133	0.61	0.0033

Con los resultados obtenidos en la figura de arriba, se verifica estadísticamente la tabla proveída en la consigna.

Para el siguiente ejercicio se genera la misma cantidad de patrones pero entre sí tendrán una correlación, o sea, serán copias de un patrón base con un porcentaje de los elementos invertidos. De esta manera se puede evaluar como distintas correlaciones afectan a la capacidad de los datos.

Para esto se ha definido un patrón base, similar a los que se utilizaron para el anterior ejercicio, con 1s y -1s seleccionados al azar. Luego se copia y dependiendo de la correlación se hará alterarán una determinada cantidad de elementos. Es decir, si entre sí tienen una correlación de 0.9 con 300 elementos, habrán 30 elementos distintos.

Para visualizar el efecto de la correlación, se hace un gráfico comparativo entre la cota de error y la cantidad de patrones aprendidos para cada correlación.



Como se puede ver en la figura, se alcanza mayor cantidad de patrones aprendidos al tener una correlación 0.5. Se puede asumir que esto se debe a que los valores son seleccionados de manera pseudo-aleatoria y que al cambiar la mitad de los valores no afecta sustancialmente la capacidad de la red. De la misma manera, se puede ver que para una correlación c y su complemento $1 - c$ tienen capacidades muy similares entre sí. Se puede asumir que esto se debe a la similitud entre patrones haciendo que la red no supiese distinguir entre uno u otro por ende generando más error.