



SIA TP4

Gastón Lifschitz
Lucio Pagni



Ejercicio 1

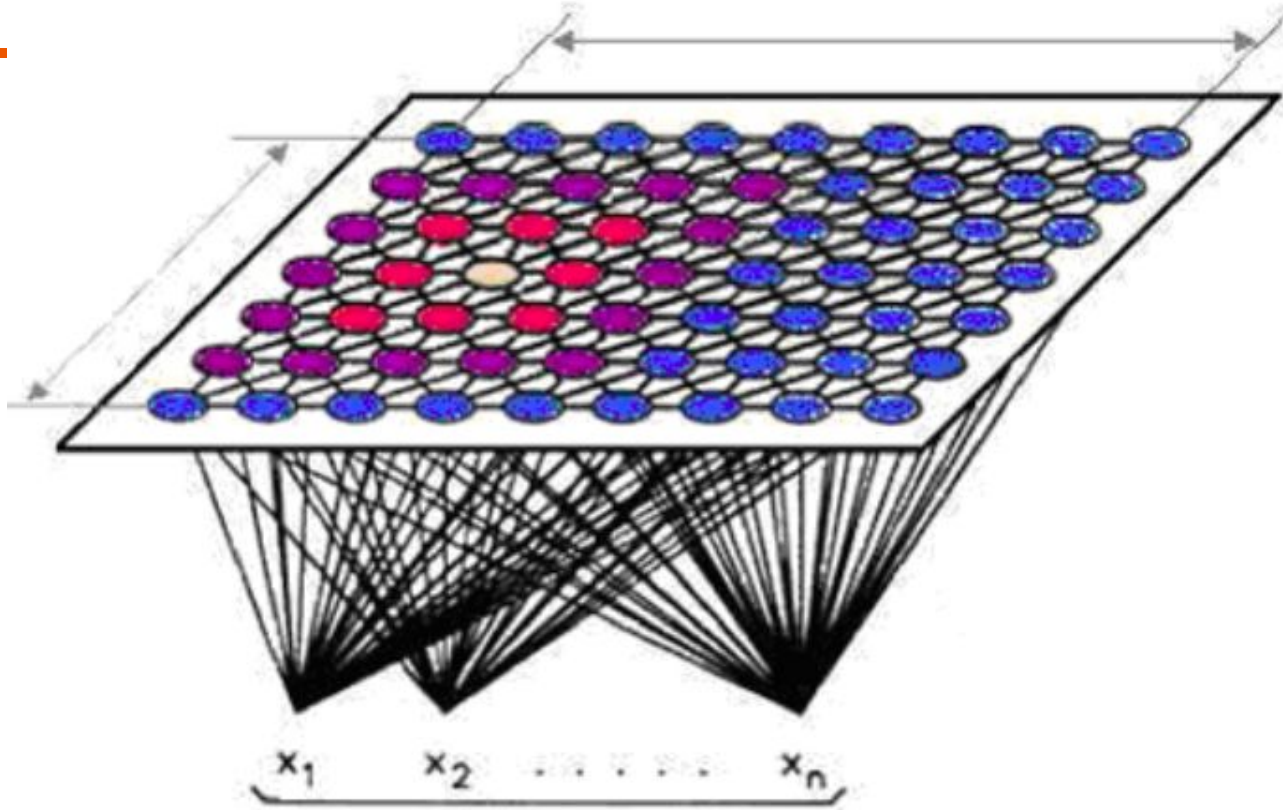
Análisis del conjunto de datos europe.csv





Ejercicio 1.a

Implementar la red de Kohonen



Asociar países que tengan las mismas características geoeconómicas y sociales



Clase 0-> Austria Belgium Denmark

Clase 1-> Luxembourg Netherlands

Clase 2-> Czech Republic

Clase 3->

Clase 4-> Greece Ireland Italy Norway Spain Switzerland United Kingdom

Clase 5-> Bulgaria Croatia Estonia Hungary Latvia Lithuania Portugal Slovakia Slovenia

Clase 6-> Finland Germany Sweden

Clase 7-> Iceland

Clase 8-> Poland Ukraine

Realizar al menos un gráfico que muestre los resultados



Bulgaria Estonia Hungary Latvia
Poland Ukraine
Czech Republic Lithuania Slovakia Slovenia
Finland Germany Italy Sweden
Denmark Netherlands Norway Switzerland
Austria Belgium Iceland
Croatia Greece Portugal Spain United Kingdom

Distancia Promedio Entre Neuronas Vecinas



- Clase 0-> Belgium
- Clase 1-> Finland
- Clase 2-> Austria Denmark Iceland Ireland
- Clase 3-> United Kingdom
- Clase 4-> Germany
- Clase 5-> Luxembourg Netherlands Norway Switzerland
- Clase 6-> Greece Portugal
- Clase 7-> Bulgaria Croatia Czech Republic Estonia Hungary Latvia Lithuania Poland Slovakia Slovenia Ukraine
- Clase 8-> Italy Spain Sweden

Analizar la Cantidad de Elementos que fueron asociados a cada neurona

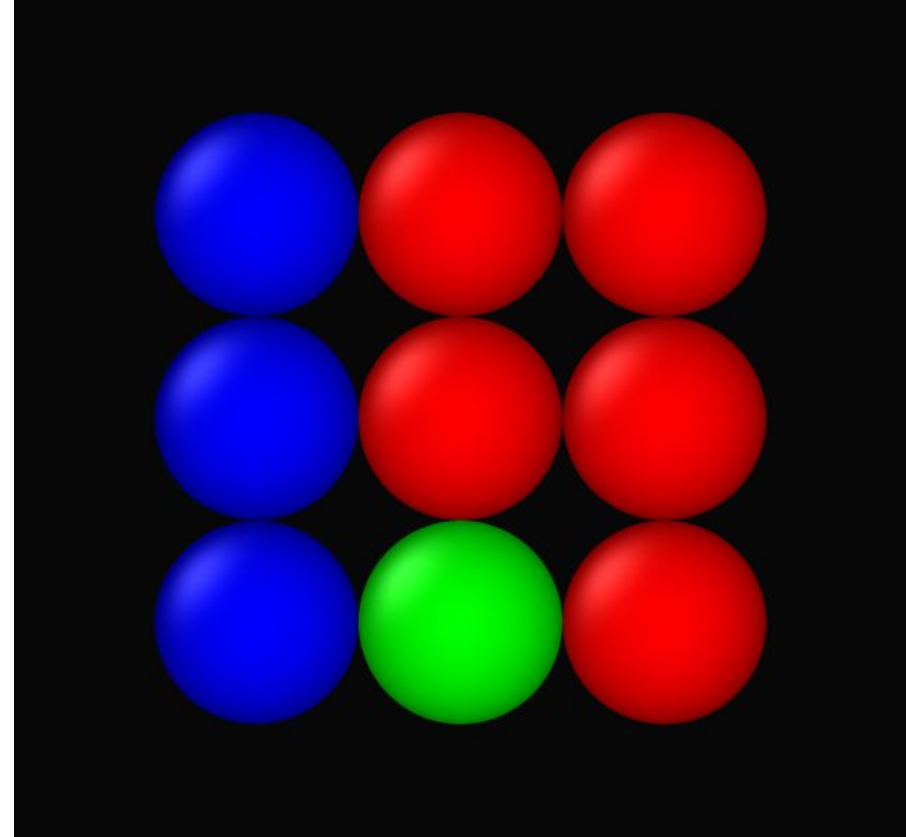
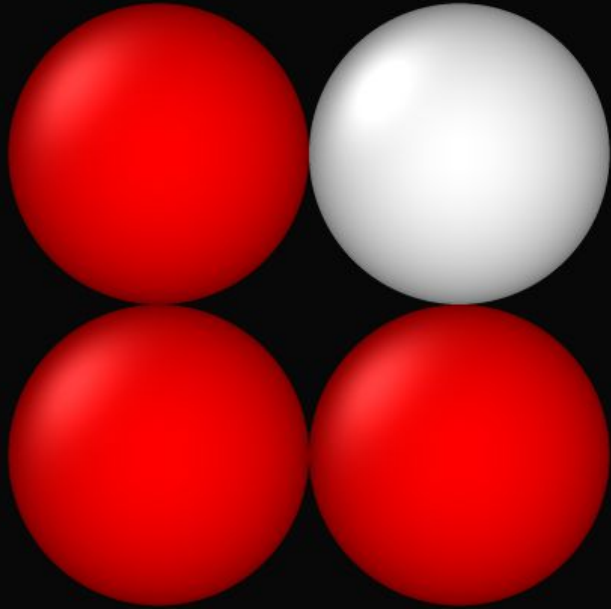
Forma Vectorial

1.00 1.00 4.00 1.00 1.00 4.00 2.00 11.00 3.00


Forma Matricial

1.0	1.0	4.0
1.0	1.0	4.0
2.0	11.0	3.0

Realizar un gráfico que muestre las distancias promedio entre neuronas vecinas



Referencias



```
if(navgd[i][j] <= 0.33) {  
    color = " 0 0 1";//Green  
}  
if(navgd[i][j] >= 0.34 && navgd[i][j] <= 0.66) {  
    color = " 0 1 0";//Blue  
}  
if(navgd[i][j] >= 0.67) {  
    color = " 1 0 0";//Red  
}
```

Está definido en una escala de 0 a 1 porque la matriz interna que representa a la distancia promedio entre nodos la estandarizamos antes de analizarla para imprimir.

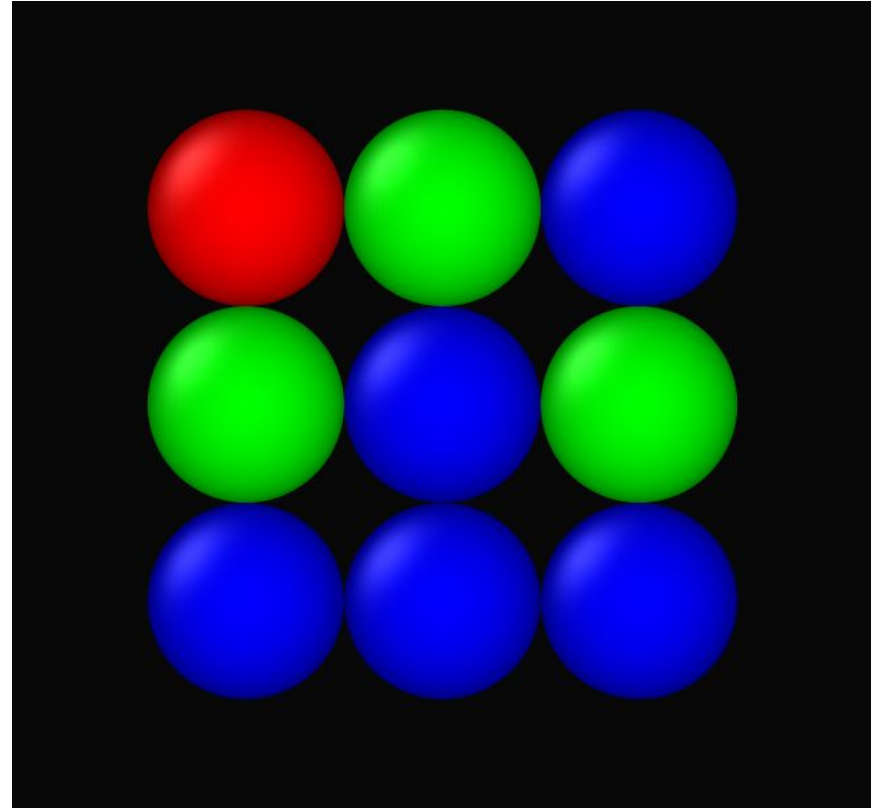
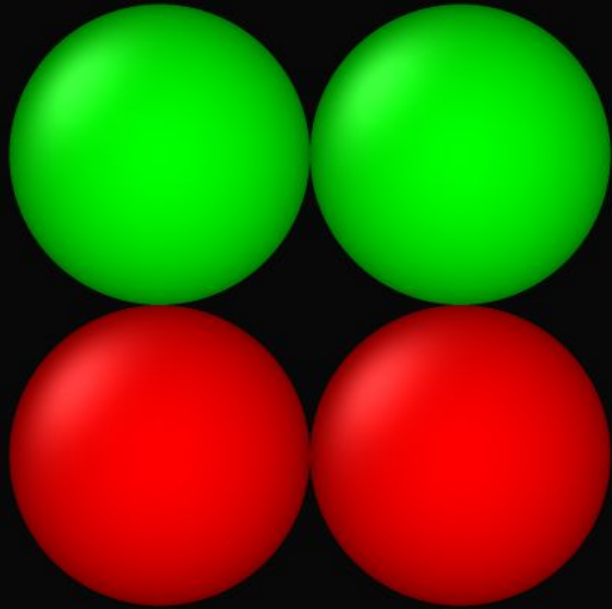
Los gráficos anteriores mostraban el estado final. Es super válido preguntarse,

Cómo es la Evolución a lo largo del entrenamiento?


Buscamos capturar la esencia del proceso de aprendizaje que básicamente las neuronas se van moviendo.

En este caso vemos que al principio hay varios cambios y luego no, en realidad si hay cambios pero al ser menores no se representan con la escala que estamos manejando, esto se regula con los hiperparametros de las funciones tasa de aprendizaje y sigma.

Analizar la cantidad de elementos que fueron asociados a cada neurona



Referencias



```
if(count[id] <= 3) { //Blue
    color = "0 0 1";
}
if(count[id] >= 4 && count[id] <= 7) { //Green
    color = "0 1 0";
}
if(count[id] >= 8) { //Red
    color = "1 0 0";
}
```

Los rangos definidos en este caso se definen para marcar que


ROJO tiene “muchos” países

VERDE tiene una cantidad “intermedia” de países

AZUL tiene “pocos” países

Qué seguridad tenemos en el código escrito?

Los test unitarios aportan cierto grado de seguridad en el código escrito. Nos aseguran que los métodos de la clase Kohonen están haciendo lo que tienen que hacer correctamente.



Los resultados "estables" que obtenemos al correr el programa tienen regularidad. Llamamos estables a los resultados que tiene una cantidad similar de resultados por clase, ignorando las clases vacías. Esto es debido a que como inicializamos valores al azar, puede ocurrir que las distancias iniciales hagan que algunos inputs queden muy lejos. Sumado a esto, está la aleatoriedad del proceso de entrenamiento el cual también toma tuplas al azar.

Clase 0-> Bulgaria Estonia Hungary Latvia Poland Ukraine

Clase 1->

Clase 2-> Czech Republic Lithuania Slovakia Slovenia

Clase 3-> Finland Germany Italy Sweden

Clase 4-> Ireland Luxembourg

Clase 5->

Clase 6-> Denmark Netherlands Norway Switzerland

Clase 7-> Austria Belgium Iceland

Clase 8-> Croatia Greece Portugal Spain United Kingdom

Resultado Estable

Clase 0->

Clase 1-> Switzerland 

Clase 2-> Finland Germany Norway Spain Sweden

Clase 3-> Italy

Clase 4-> Ireland Luxembourg United Kingdom

Clase 5-> Iceland

Clase 6-> Bulgaria Estonia Hungary Latvia Poland Ukraine

Clase 7-> Croatia Greece Portugal

Clase 8-> Austria Belgium Czech Republic Denmark Lithuania Netherlands Slovakia Slovenia

Resultado Instable

Clase 0->

Clase 1->

Clase 2->

Clase 3->

Clase 4->

Clase 5-> Ukraine

Clase 6-> Austria Belgium Bulgaria Croatia Czech Republic Denmark Estonia Finland Germany Greece Hungary Iceland Ireland Italy Latvia Lithuania Luxembourg Netherlands Norway Poland Portugal Slovakia Slovenia Spain Sweden Switzerland United Kingdom

Clase 7->

Clase 8->



Ejercicio 1.b

Implementar la regla de Oja

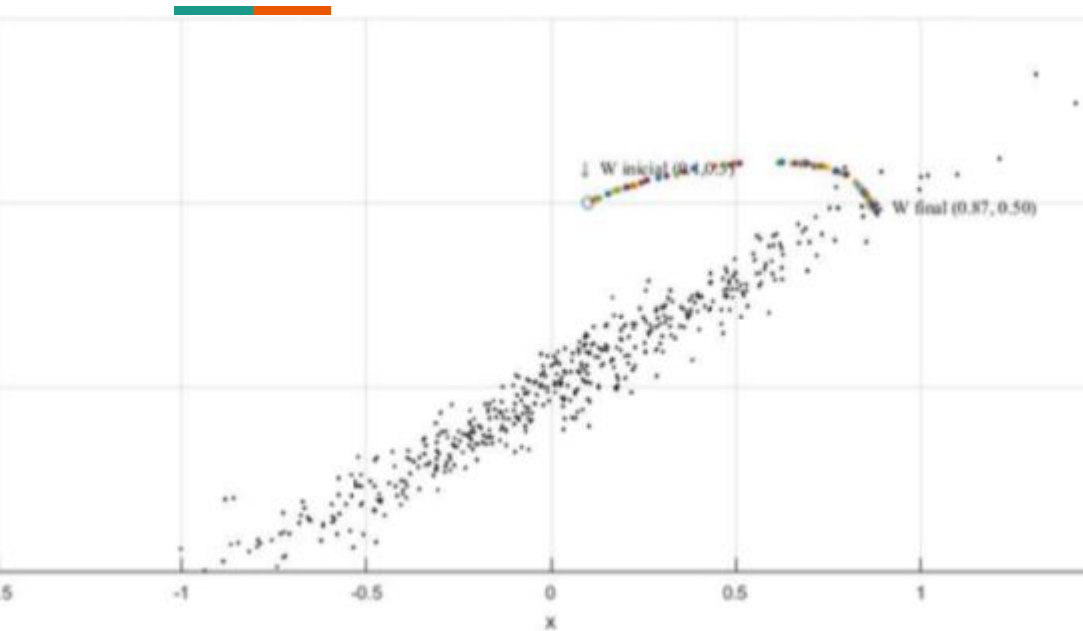


Diagram illustrating the implementation of Oja's rule:

Inputs: \bar{x}_i (examples) and $\text{attn} \begin{bmatrix} x_0 & x_1 & x_2 \end{bmatrix}$ (Neurons).

Weights: $\text{attn} \begin{bmatrix} w_0 & w_1 & w_2 \end{bmatrix}$ (Neurons).

Weight vector: \bar{w}_i .

Calculation of the dot product:

$$y = \bar{x}_i \cdot \bar{w}_i = \sum_{i=1}^n \bar{x}_i \cdot \bar{w}_i$$

Calculation of the weight update:

$$\Delta \bar{w}_i = \eta \bar{x}_i * y = \eta \bar{x}_i * \bar{w}_i$$

$$\bar{w}_i^{k+1} = \bar{w}_i^k + \eta \bar{x}_i * \bar{w}_i$$

Oja's rule:

$$\Delta \bar{w}_i = \eta \left(\bar{x}_i * \bar{y} - \frac{\bar{y}^2}{2} * \bar{w}_i \right)$$

Initial weight vector:

$$\bar{w}_0 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Condition for the initial weight vector:

$$\bar{w}_0 \leq \frac{1}{2} \quad \bar{w}_0 = 0,5$$

Calcular la primer componente principal de europe.csv



Al calcularlo con nuestra implementación obtenemos

$[-0.13, 0.50, -0.41, 0.48, -0.18, 0.47, -0.27]$

Y a veces obtenemos

$[0.13, -0.50, 0.41, -0.48, 0.18, -0.47, 0.27]$

Interpretar el resultado de la primer componente



Las componentes principales son utilizadas para extraer características destacadas o importadas de un conjunto de datos bajando la dimensión del mismo.

La componente Y1 nos permite agrupar los registros que tengan valores de Y1 parecidos.

O también ordenar registros de acuerdo a esta característica.

Con la regla de Oja podemos calcular la primer componente (si queremos calcula el resto deberíamos usar la regla de Sanger).

Comparar el resultado del ej. 1b2 con el resultado de usar una librería

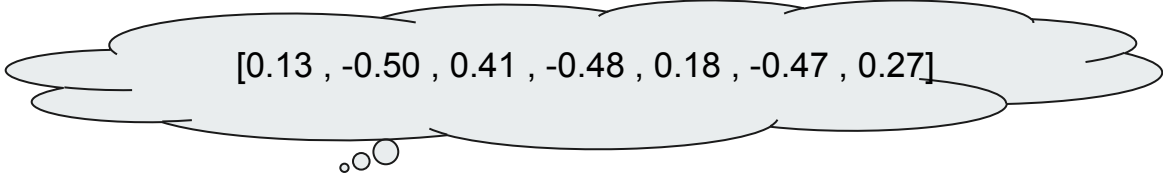


Al calcularlo con una librería obtenemos

[0.1248739 , -0.50050586 , 0.40651815 , -0.48287333 , 0.18811162 , -0.47570355 , 0.27165582]

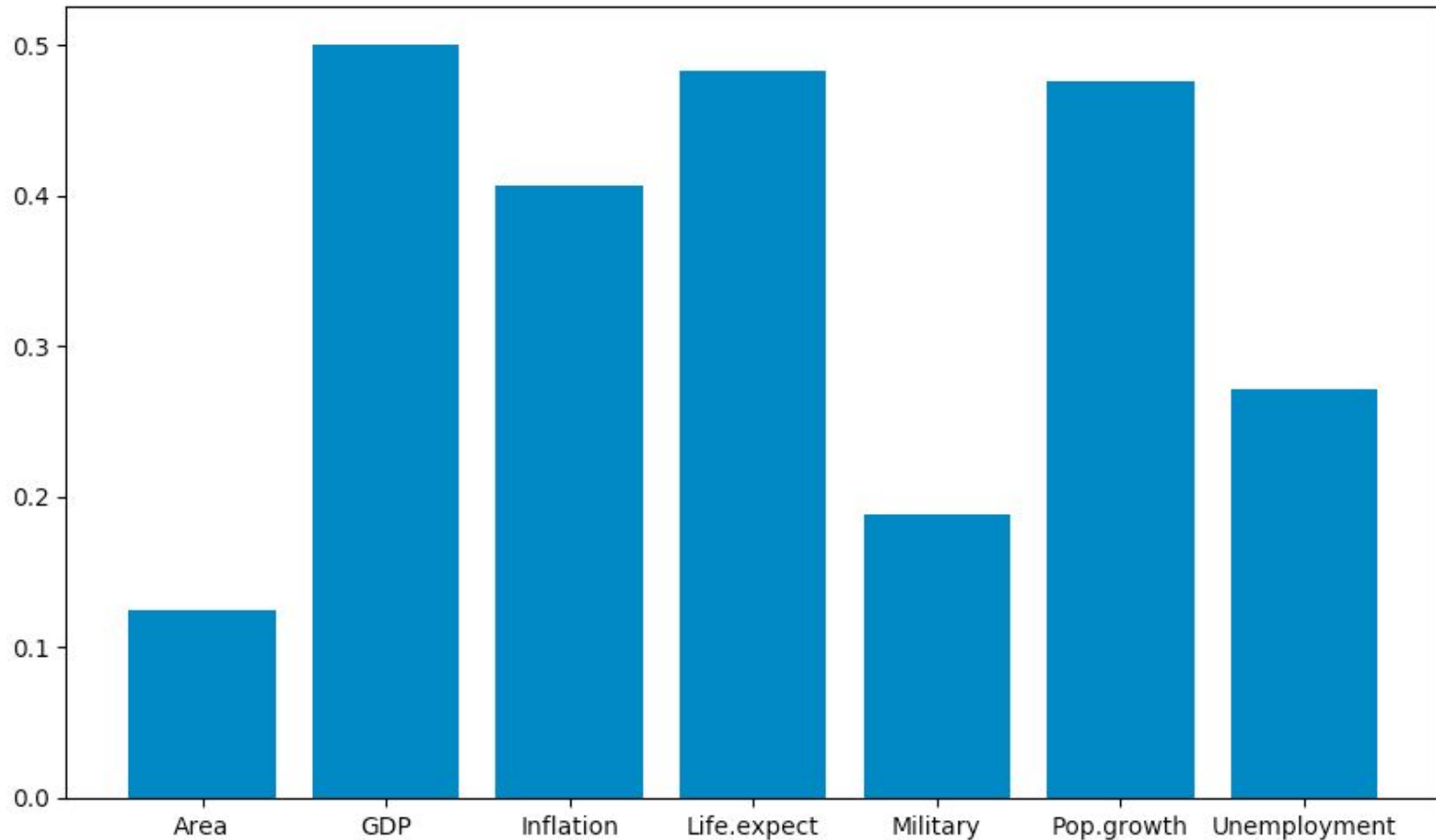
Obtenemos el resultado como vectores con la misma dirección pero con sentido opuesto, concluimos que los resultados son equivalentes. El signo puede variar según el learning rate.

Tuvimos que poner un learning rate de 0.001 , es decir, un valor muy pequeño para evitar resultados tales como Infinity y NaN



[0.13 , -0.50 , 0.41 , -0.48 , 0.18 , -0.47 , 0.27]

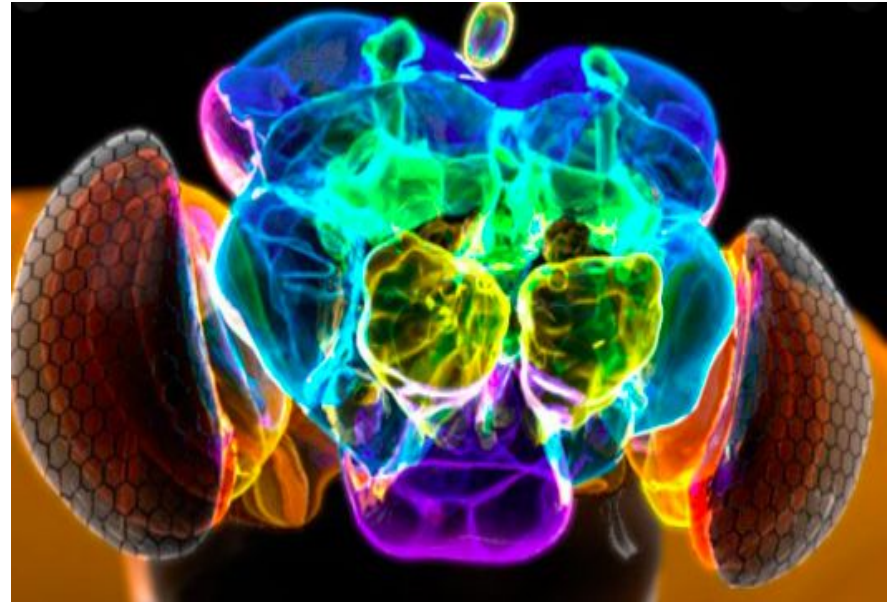
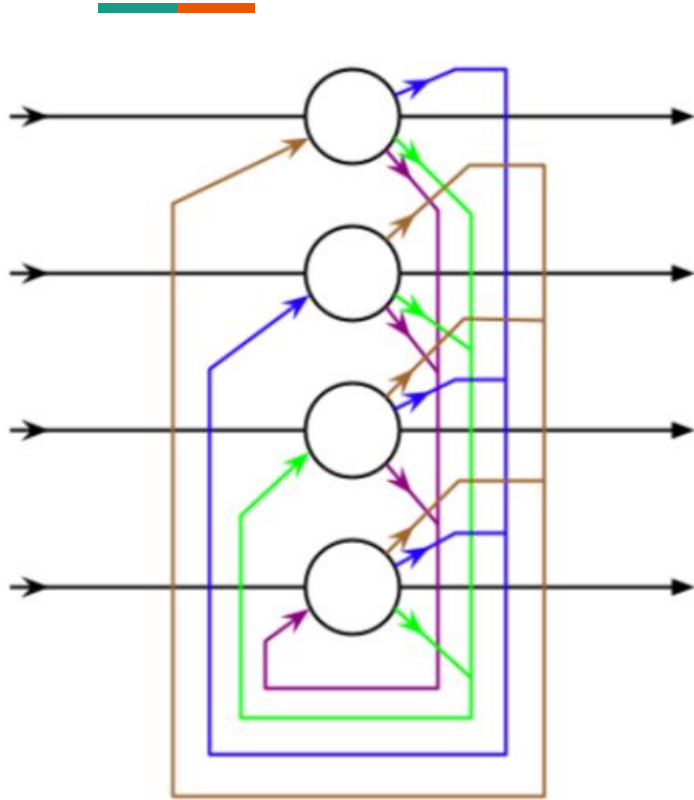
Contribución de cada atributo a la primer componente principal





Ejercicio 2

Aplicar el modelo de Hopfield para asociar matrices ruidosas con patrones almacenados



Patrones almacenados (metodo train)



Letras A, J, C y M (todas en mayúscula)

Corremos la red con un pequeño cambio (método run)

Input:

```
* * * * *  
      *  
      *  
    * *  
* * *
```

Output:

```
* * * * *  
      *  
      *  
    * *  
*      *  
* * *
```

Input:

```
*      *  
* *    * *  
* *    *  
*      *  
*      *
```

Output:

```
*      *  
* *    * *  
*      *  
*      *  
*      *
```

Input:

```
* *    * *  
* *  
*  
* * * * *
```

Output:

```
* * * * *  
*  
*  
*  
* * * * *
```

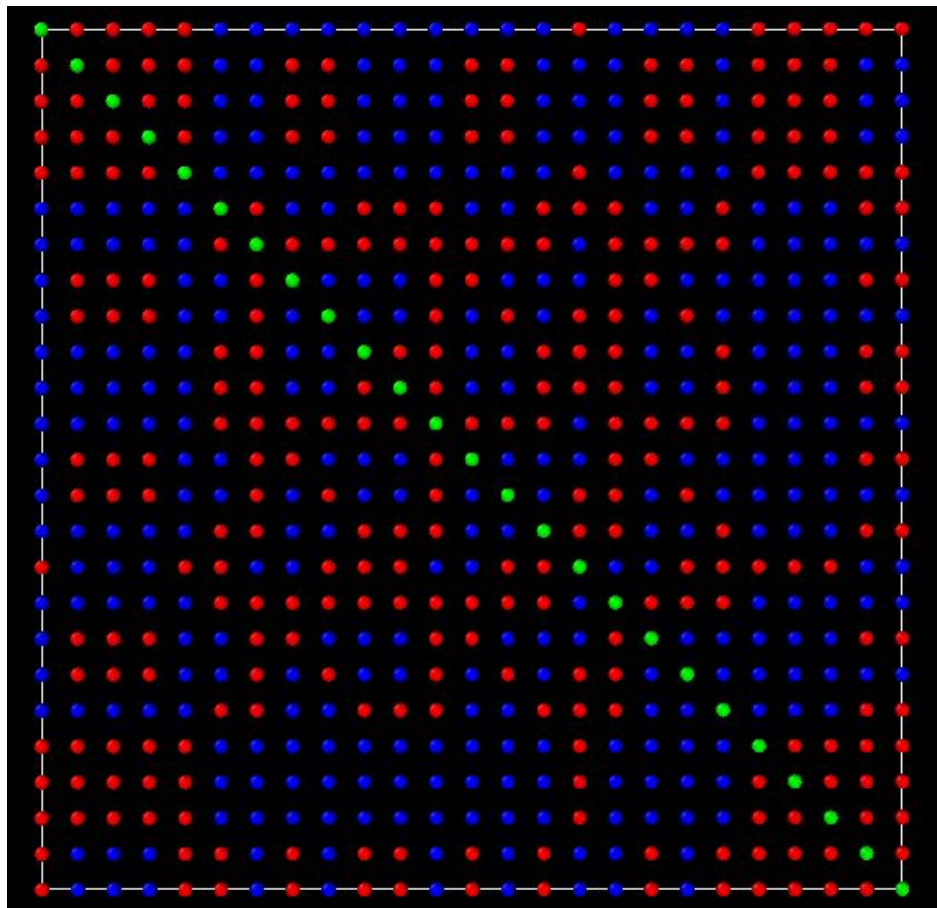
Input:

```
      *  
    * *  
  * * *  
    * *  
*      *
```


Output:

```
      *  
    * *  
  * * *  
    * *  
    * *
```


Matriz de pesos



Ingresar un patrón ruidoso e identificar un estado espurio



```
Input:
* * * * *
* * * * *
* * * * *
* * * * *
* * *   *

Output:
*       *
*   *   *
*       *
*   *   *
*       *
```

```
Input:
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *

Output:
*       *
*   *   *
*       *
*   *   *
*       *
```


- Ambos patrones de entradas son ruidos y muy similares entre sí
- Producen la misma salida, generando así un estado espurio.



- Al ingresar una letra que es parecida a una entrenada, podemos ver como la *i* mayúscula la convierte en *C* ya que tiene pocas alteraciones (solo 6 posiciones de 625 cambian).



Conclusiones



La red de Kohonen aprende la distribución de los datos de entrada. La cantidad de neuronas tendría que ser similar a la cantidad de clases que espero obtener. Inicializar los pesos al azar puede generar inestabilidad en los resultados obtenidos. Debido a el problema de las unidades muertas.

La regla de Oja calcula la primer componente principal, a veces puede dar en la misma dirección pero en sentido contrario.

La red de Hopfield puede tener mínimos locales los cuales no son los patrones almacenados , pero al ser mínimos locales, también son atractores. Se conocen como estados espurios. Las memorias asociativas de Hopfield tienen una fuerte limitación en la cantidad de patrones que puede almacenar.

Curiosidad: La primer componente tiene un peso de 0.46102367%