DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

# Trabajo Práctico II

Redes Neuronales

Primer Cuatrimestre de 2022


| Integrante | LU | Correo electrónico |
|---|---|---|
| Laura Valeria Rodriguez | 224/14 | lvrproduccion@gmail.com |
| Valentín Franco Valls | 449/19 | valentinvalls@hotmail.com |
| Agustín Delmagro | 596/14 | agustin.delmagro@gmail.com |






Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2160 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (54 11) 4576-3359

http://www.fcen.uba.ar

# 1. Reducción de dimensiones

Nuestra implementación del *aprendizaje hebbiano* cuenta con 850 unidades de entrada coincidentes con la dimensión de los datos del *Bag of Words* y 9 unidades de salida correspondiéndose con las dimensiones a las que queremos reducir los datos.

Los datos fueron estandarizados con media 0 y varianza 1 ya que es un requisito para la convergencia del *aprendizaje hebbiano*.

Para la actualización de los pesos decidimos utilizar las reglas de **Oja** y **Sanger** y no la regla de **Hebb** ya que no nos pareció apropiada para tratar este problema.

Utilizamos 2 condiciones de corte para el algoritmo, un límite de épocas y la ortogonalidad de los pesos entre las unidades de salida que indica que se encontró un subespacio de dimensiones menores donde se pueden representar los datos conservando más información.

Por lo general las reglas que utilizamos tienden a converger a una solución. La modificación de hiperparámetros como el *learning rate*, la inicialización de los pesos y las condiciones de corte deberían influir principalmente en la cantidad de épocas que se toma en conseguir la solución esperada.

Para graficar los resultados obtenidos de la reducción de dimensiones tomamos los features de cada instancia de salida en pares de 3 representándolos como un punto de $R^3$ en 3 gráficos distintos. Además etiquetamos cada uno de estos puntos con la categoría del documento a la que pertenece para que cada punto tenga un color que se corresponda con el tipo de documento que es. Esto nos permitirá ver la forma en que se estén agrupando los datos y si efectivamente los algoritmos logran separarlos en clusters.

Para la **regla de Oja** obtuvimos el mejor resultado al ejecutar el algoritmo por 500 épocas y con un learning rate adaptativo de 0,0001/época. En la figura 1 podemos ver que para las 3 componentes principales $Y_1$, $Y_2$ y $Y_3$ se pueden ver unos 7 clústers mientras que para el resto de las componentes los datos se encuentran mucho más mezclados no pudiendo distinguir más de 5 clusters. Probablemente esto se deba a que **Oja** está pensado para extraer solamente la primer componente principal lo cual explique que le resulte más difícil distinguir los datos utilizando el resto de las componentes principales. Con una menor cantidad de épocas los datos aún no fueron suficientemente separados por lo que se aprecian menos clústers y con una mayor cantidad de épocas aquellos datos que ya estaban clusterizados se separan mucho más pero en ningún caso podemos distinguir una mayor cantidad de clústers por lo que consideramos este valor óptimo. En cuanto al learning rate su modificación influyó principalmente en una mayor cantidad de épocas para alcanzar una cantidad de clústers similares a los de la solución descripta anteriormente.

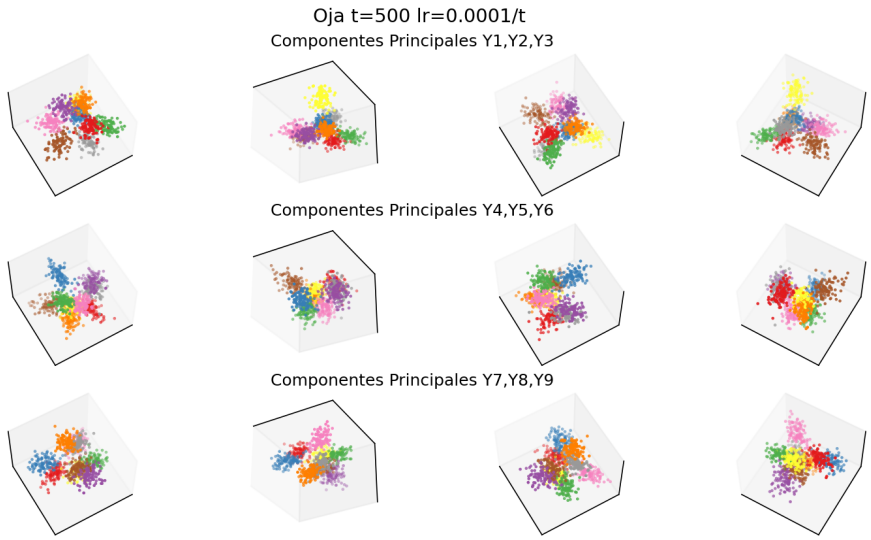



Figura 1: Representación de puntos obtenidos con regla de Oja

En cuanto a la **regla de Sanger** obtuvimos una solución óptima con los mismos parámetros que utilizamos en Oja. Sin embargo, tal como podemos observar en la figura 2 los datos fueron separados de mejor forma ya que para las 3 componentes principales $Y_1$, $Y_2$ y $Y_3$ podemos ver unos 8 clústers además de que para el resto de las componentes podemos ver entre 6 y 7 clústers. Dado que Sanger es una generalización de Oja a múltiples salidas es esperable que performe mejor que Oja al buscar más de una componente principal. Así mismo también se puede apreciar que aquellos datos que pertenecen a un mismo clúster se encuentran más próximos entre sí que en el caso de Oja. Al igual que en el caso anterior una menor cantidad de épocas no separa suficientemente los datos mientras que una mayor cantidad de épocas no permite distinguir más clusters sino que solamente separa más los datos ya clusterizados.

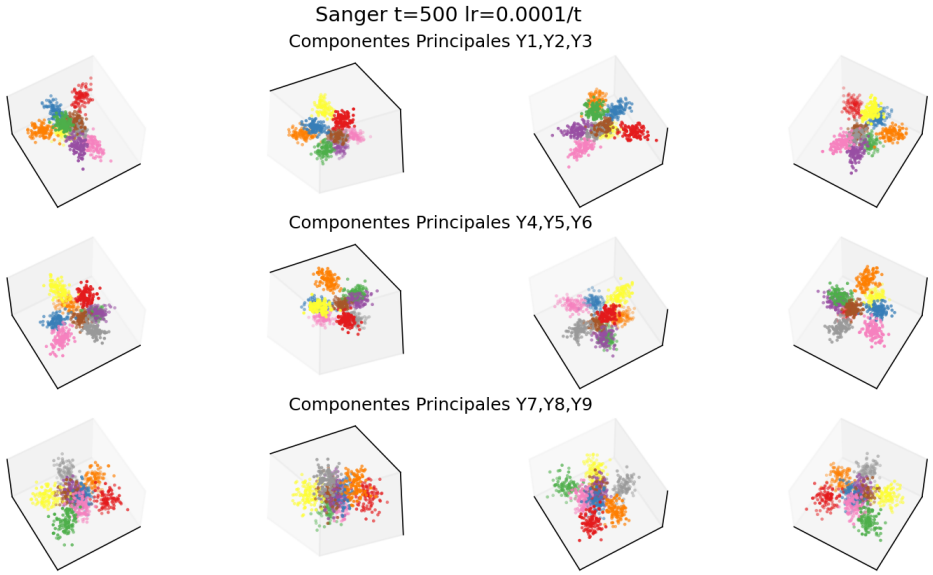



Figura 2: Representación de puntos obtenidos con regla de Sanger

## 2. Mapeo de características

El modelo de mapeo de características auto-organizado que implementamos sigue lo visto en clase, en este caso para representar las neuronas utilizamos un tensor de MxMxN donde N es la cantidad de unidades de entrada (en nuestro caso 850) y MxM es el tamaño de la grilla de salida.

La idea es clasificar automáticamente los documentos dentro de las 9 categorías establecidas por lo que estimamos que el mínimo valor de M será 3 obteniendo una grilla de salida de 3x3. Tener mayor dimension podría mejorar la categorización, pero se debe considerar que valores muy grandes de M pueden aumentar innecesariamente el tiempo de ejecución y entregar una grilla de salida con muchas neuronas muertas (es decir, neuronas que no mapean a ninguna categoría).

Separamos el conjunto de datos en datos de entrenamiento y datos de validación, tomando ejemplos de manera tal de mantener la uniformidad de la distribución de los datos originales. Como la amplitud de los parámetros podría estar en diferentes escalas, se realizo un proceso previo de normalización de datos para que todas las variables resulten igualmente importantes en el resultado del entrenamiento.

Se realizo una búsqueda de hiperparámetros contemplando los siguientes: tamaño de la grilla de salida, **learning rate**, **radio**, cantidad de épocas de entrenamiento, porcentaje de datos separados para validación. El *learning rate decay* y *radius decay* se fijaron en 0.1.

Una vez finalizado el entrenamiento y dado que tenemos los valores de las categorías a las que corresponde cada documento evaluado, asignamos a cada neurona de la grilla de salida el valor de la categoría que cayó predominantemente en ella.

Calculamos una medida de *accuracy* como la cantidad de documentos que fueron categorizados correctamente sobre el total de documentos evaluados. Además, finalizado el entrenamiento, calculamos para cada documento del conjunto de validación su Unidad de Mejor Correspondencia dentro de la grilla, asignándole la categoría que correspondía a esa posición y calculando luego el *accuracy* de este conjunto.

Repetimos esto para cada una de las posibles combinaciones de hiperparámetros quedándonos finalmente con la que mejor *accuracy* obtuvo sobre el conjunto de validación. En el cuadro 1 se pueden ver algunos de los valores obtenidos.

| lr | radio | m | val | epochs | train_acc | val_acc |
|---|---|---|---|---|---|---|
| 0.01 | 1 | 3 | 0.10 | 8 | 0.678133 | 0.674419 |
| 0.3 | 3 | 6 | 0.20 | 24 | 1.000000 | 1.000000 |
| 0.01 | 1 | 9 | 0.10 | 16 | 1.000000 | 0.965116 |
| 1 | 10 | 9 | 0.10 | 16 | 0.954545 | 0.930233 |
| 0.01 | 1 | 8 | 0.20 | 40 | 1.000000 | 0.870056 |
| 0.75 | 3 | 8 | 0.20 | 40 | 0.993084 | 1.000000 |

Cuadro 1

Tal como indicaba la bibliografía sugerida por la cátedra pudimos observar que valores pequeños del radio provocan que solo se modifique el peso de células individuales mientras que valores grandes (y en particular cuando el M utilizado es chico), pueden terminar afectando a todas las células perdiendo la idea de vecindad. Por otra parte, valores muy pequeños del *learning rate* hacen innecesariamente lento el aprendizaje (o practicamente nulo, segun la cantidad de épocas y valor del *learning rate decay* utilizado).

Finalmente como se puede observar llegamos en varias configuraciones a valores de *accuracy* similares y muy altos. Donde si bien los mapas quedan conformados de distinta forma en cada caso, todos realizan una excelente categorización.

Como ejemplo, en la figura 3 podemos ver como se van acomodando las categorías por las neuronas de salida en las distintas etapas del entrenamiento para el mejor modelo entrenado. Hay algunos epochs donde figura una categoria 0, simboliza las neuronas que no mapean ninguna categoría. Mientras que en las figuras 4a y 4b podemos ver como resultaron mapeados perfectamente los documentos de los sets de validación.

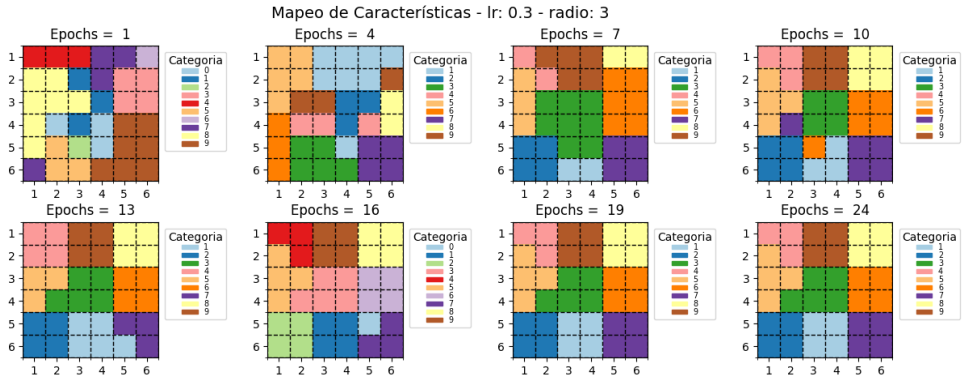
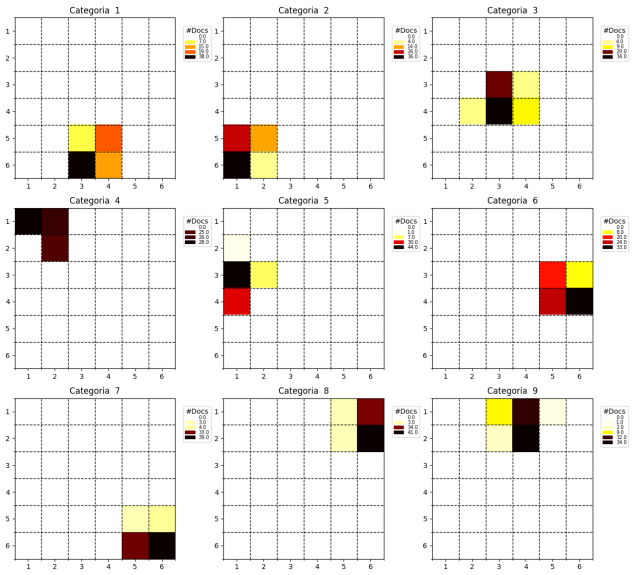
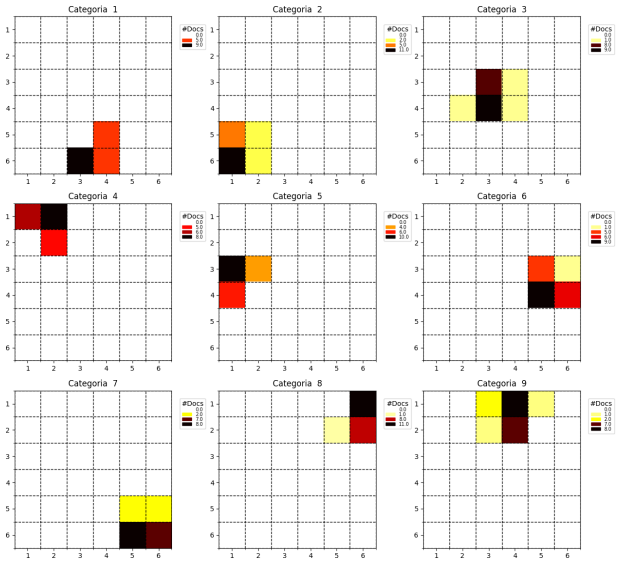



Figura 3




(a) Categorización de los datos de Entrenamiento




(b) Categorización de los datos de validación

# 3. Código entregado

Con el código entregado, brindamos un *readme.md* donde se indica claramente cómo correrlo, qué parametros recibe y qué hacen. Igualmente aquí damos unos ejemplos de uso.

- Entrenar un modelo hebbiano con regla de Oja : *python -m run –model hebb -s –args 850 9 oja*
- Levantar el mejor modelo hebbiano entrenado con Oja y predecir con el: *python -m run –model hebb –modelo_file hebb_OJA__modelo_entrenado.txt*
- Entrenar un modelo hebbiano con regla de Sanger : *python -m run –model hebb -s –args 850 9 sanger*
- Levantar el mejor modelo hebbiano entrenado con Sanger y predecir con el: *python -m run –model hebb –modelo_file hebb_SANGER__modelo_entrenado.txt*
- Entrenar un nuevo SOM: *python -m run –model som -s –args 850 6*
- Levantar el mejor modelo SOM ya entrenado y predecir con el: *python -m run –model som –modelo_file som_modelo_entrenado.txt*