Trabajo Práctico 2: Redes Neuronales Artificiales

L. Alvarez, P. Bordon y D. Santos

31 de Octubre de 2016

Índice

1. Descripción del problema

Se cuenta con documentos de descripción de empresas. Dichos documentos se encuentran preprocesados, de manera tal que se tiene un dataset que especifica qué cantidad de veces aparece cada palabra en cada documento. Este formato es conocido como Bolsa de Palabras (Bag-Of-Words). El problema a resolver será diseñar y entrenar una red neuronal mediante métodos de aprendizaje hebbiano no supervisado para que pueda clasificar automáticamente los documentos según categoría. También se busca que esta red pueda generalizar corréctamente y responda de manera certera a nuevos documentos.

Se utilizaron dos subparadigmas clásicos de aprendizaje no supervisado: aprendizaje hebbiano y aprendizaje competitivo. Se trabajó en un modelo de reducción de dimensiones y otro de mapeo de características autoorganizado para su aplicación sobre las datos propuestos. En el primer caso se planteó una reducción a tres dimensiones utilizando los algoritmos de Oja y Sanger. En el segundo se planteó un modelo de autoorganización a partir del algoritmo de Kohonen. Distintas configuraciones de parámetros fueron estudiadas en cada caso. Se presentan aquí los resultados obtenidos.

2. Ejercicio 1: Reducción de dimensiones

2.1. Introducción

Para el primer ejercicio, queremos reducir la dimensionalidad de la entrada, debido a que es muy grande, a 3 valores. Para esto vamos a diseñar una red neuronal y la entrenaremos mediante aprendizaje hebbiano no supervisado aplicando la regla de oja y la regla de sanger. Luego analizaremos y compararemos sus resultados.

2.2. Análisis de la Red

En un comienzo, trabajamos con una red que contaba con 10 o 15 neuronas de salida para no reducir drásticamente su dimensión. Al final el entrenamiento, reducíamos la salida a 3 valores y obteníamos lo buscado. Luego de varias pruebas y cambios determinamos que no ganábamos nada tomando esta determinación, por el contrario, se áumentaba la complejidad del algoritmo y se perdía tiempo revisando la ortogonalidad de las 10 columnas de la matriz de pesos, mientras que los resultados obtenidos no eran mejores. Por esto se determinó manejar desde un inicio 3 dimensiones de salida.

Vale aclarar que las reglas de oja y sanger son formas de calcular el cambio que se debe hacer a la matriz de pesos para entrenarla. Por esto, ambos modelos son iguales, solo que se aplican distintos cálculos para cada tipo de regla.

2.3. Procesamiento de Datos

Sabemos que los valores de entrada son enteros positivos, ya que determinan la cantidad de apariciones de una palabra en un texto. Y al estar la entrada preprocesada y sin las palabras mas comunes (con mas apariciones), los valores de entrada se encuentran acotados. Podríamos no haber implementado ningún tipo de preprocesamiento, pero preferimos estandarizar la entrada, de manera que los atributos presenten media cero y varianza uno, moviendose en un rango de valores similar.

2.4. Detalles de Ejecución

La implementación de la solución fue desarrollada usando el lenguaje python, con la librería numpy para facilitar las operaciones aritméticas y los gráficos fueron construidos con la librería matplotlib.pyplot.

Se implementó una clase llamada *hebbian*, que contiene las siguientes propiedades:

- **tolerancia_error**: valor que sirve para terminar el entrenamiento en caso que la matriz sea suficientemente ortnonormal.
- **cantidad_epocas**: valor máximo de épocas que se entrenará la red, en caso de no salir por la tolerancia de error.
- **input_file**: archivo del dataset de entrada.
- **oja**: valor que indica si se utilizará la regla de oja o sanger para entrenar (Oja: 1; Sanger: 0).

Al correr el programa, este cargará por defecto una red entrenada con la regla de sanger. Luego, pueden ejecutarse las siguientes acciones:

- **help**: Muestra por pantalla una explicación de los comandos que se pueden utilizar.

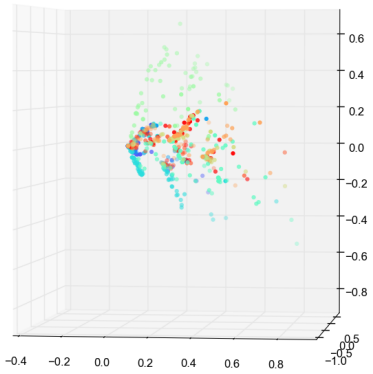
- **exit**: Termina la ejecución del programa.
- **train**: Comienza el entrenamiento de la red.
- **change**: Sirve para modificar el valor de una propiedad.
- **export**: Exporta la red a un archivo de texto.
- **import**: Importa una red desde un archivo de texto.
- **graph train**: Grafica los errores en la ortonormalidad de la matriz de pesos obtenidos durante el entrenamiento.
- **show color**: Muestra gráficos de los resultados diferenciados por categoría.
- **show train**: Muestra gráficos de los resultados de entrenamiento.
- **show valid**: Muestra gráficos de los resultados de validación.
- **varianza**: Muestra un gráfico sobre la varianza de los datos de entrada de cada categoría.

2.5. Experimentación y Resultados

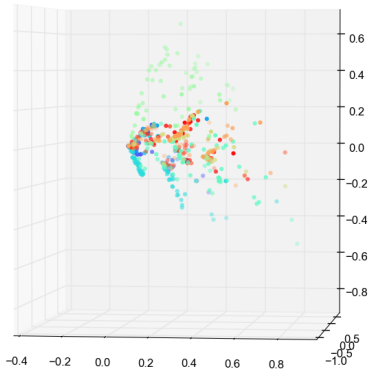
Habiendo fijado el número de neuronas de salida a tres unidades, pasamos a implementar los dos algoritmos propuestos. En primera instancia trabajamos en una fase de entrenamiento, donde el parámetro de interés a optimizar es el coeficiente de aprendizaje, que bien puede ser función del número de época. Tanto la condición de ortonormalidad($||W^T \cdot W - I|| = 0$) como la subsiguiente validación son las herramientas para determinar su valor óptimo. Comenzamos con un coeficiente de aprendizaje constante, siguiendo con funciones del tipo $\frac{1}{epoca^{\alpha}}$, con alfa entre 0 y 1. Probando estas distintas opciones encontramos que si bien no existían diferencias significativas, un coeficiente del tipo $\frac{1}{epoca^{\frac{1}{2}}}$ presentaba los mejores resultados.

2.5.1. Algoritmo de Oja

A continuación se pueden observar los resultados obtenidos utilizando el algoritmo de Oja separando el set de datos de entrada en 70 % - 30 %, para el coeficiente de aprendizaje adaptativo propuesto. Las figuras 1 y 2 presentan los datos en el espacio 3d de salida obtenido por el algoritmo. Las figuras 1a y 2a corresponden a datos de entrenamiento, mientras que las figuras 1b y 2b corresponden a datos de validación.

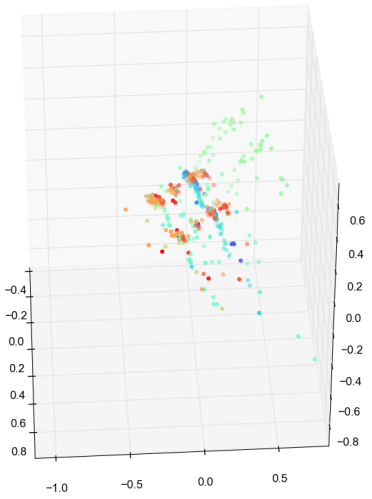


(a) Entrenamiento

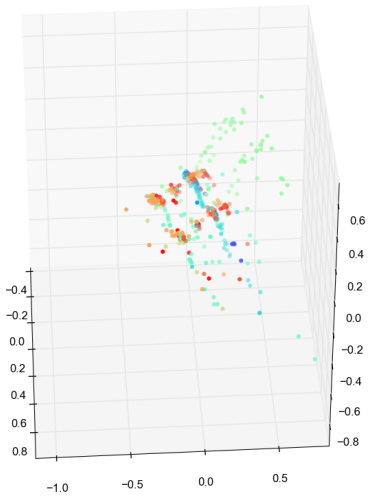


(b) Validacion

Figura 1: Vista 1 en 3d de resultados de validación y entrenamiento



(a) Entrenamiento

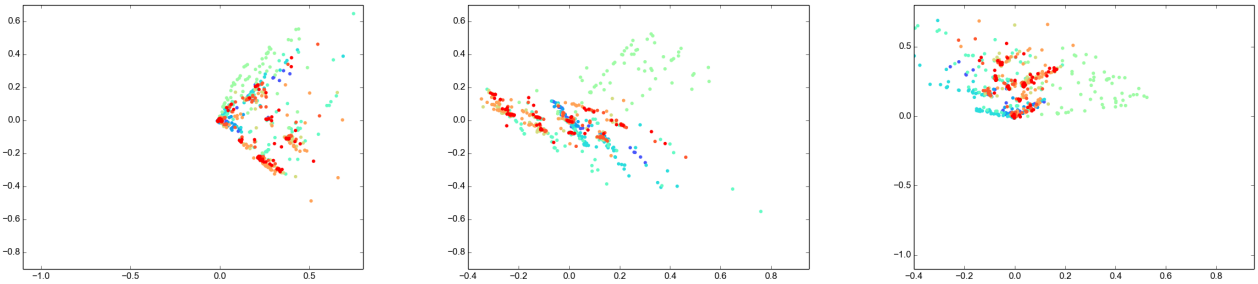


(b) Validacion

Figura 2: Vista 2 en 3d de resultados de validación y entrenamiento

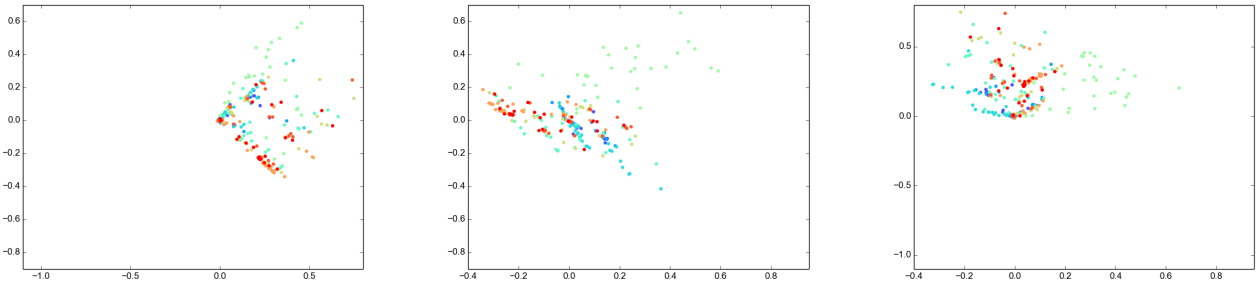
Se puede observar que la validación es consistente con la clasificación obtenida en la etapa de entrenamiento,

dando cuenta de la validez del procedimiento realizado. También se puede observar en el gráfico claramente la separación entre dos de las categorías, la verde y azul, cosa que no ocurre para las categorías restantes. Para poder discriminar sobre estas últimas realizamos graficos 2d (figura 3), proyectando el gráfico 3d sobre tres planos distintos, tanto para los datos de entrenamiento 3 como para los de validación 4.



(a) Corte 1 (b) Corte 2 (c) Corte 3

Figura 3: Cortes del gráfico de los datos de entrenamiento

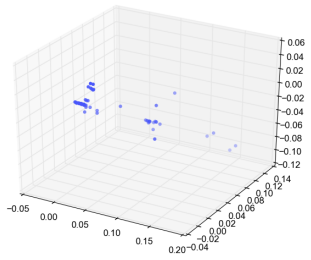


(a) Corte 1 (b) Corte 2 (c) Corte 3

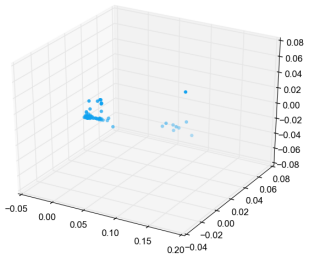
Figura 4: Cortes del gráfico de los datos de validación

Estos gráficos permiten observar mejor las similitudes entre los resultados obtenidos con los datos de entrenamiento y los obtenidos con los datos de validación. No se distinguen resultados tan errado como para ser detectados a simple vista. Esto nos deja la impresión de que, a pesar de que las categorías no se separen y agrupen como esperábamos, la red generaliza bastante bien y es consistente en cuanto a la ubicación de las categorías.

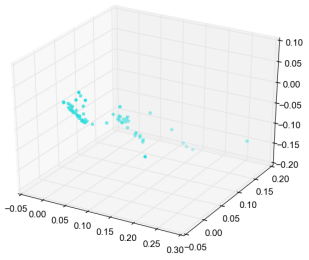
A continuación realizamos un gráfico 3D para cada categoría en la etapa de entrenamiento, con la finalidad de poder observar su distribución individualmente, ya que en las figuras 3 se encuentran algo superpuestas. Los resultados se observan en las figuras 5.



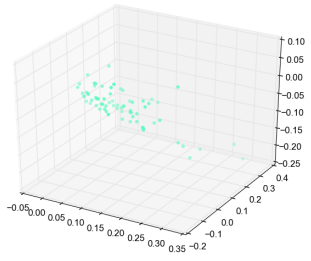
(a) Categoría 1



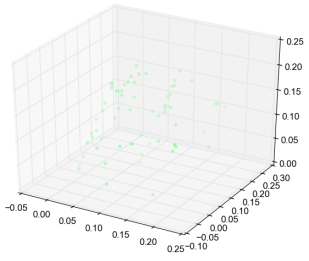
(b) Categoría 2



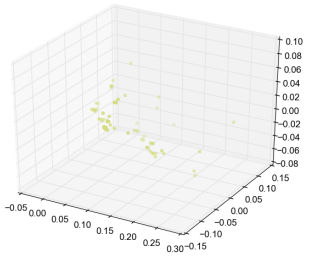
(c) Categoría 3



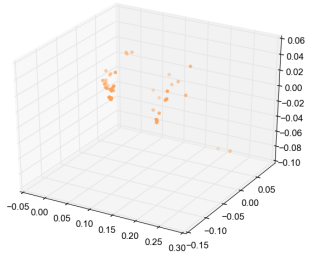
(d) Categoría 4



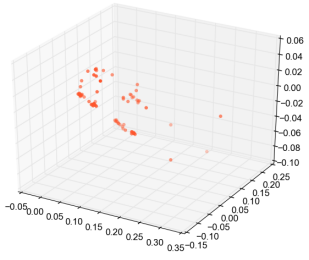
(e) Categoría 5



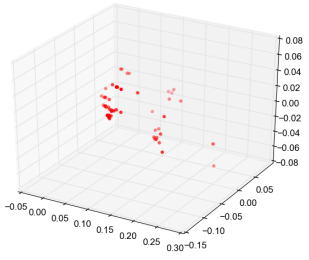
(f) Categoría 6



(g) Categoría 7



(h) Categoría 8



(i) Categoría 9

Figura 5: Gráficos de resultados diferenciados por categoría

Como observamos que algunas categorías se encontraban algo dispersas, calculamos la varianza promedio de los datos de cada categoría para ver si nos dan alguna señal de por qué puede estar ocurriendo esto. Los resultados se pueden observar en la siguiente figura 6.

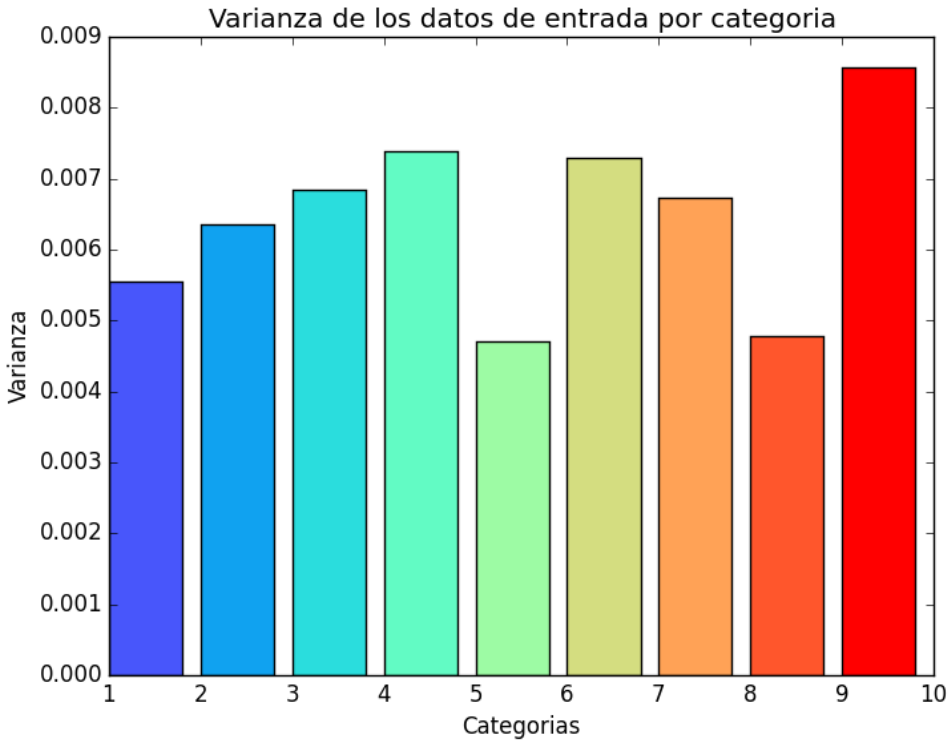
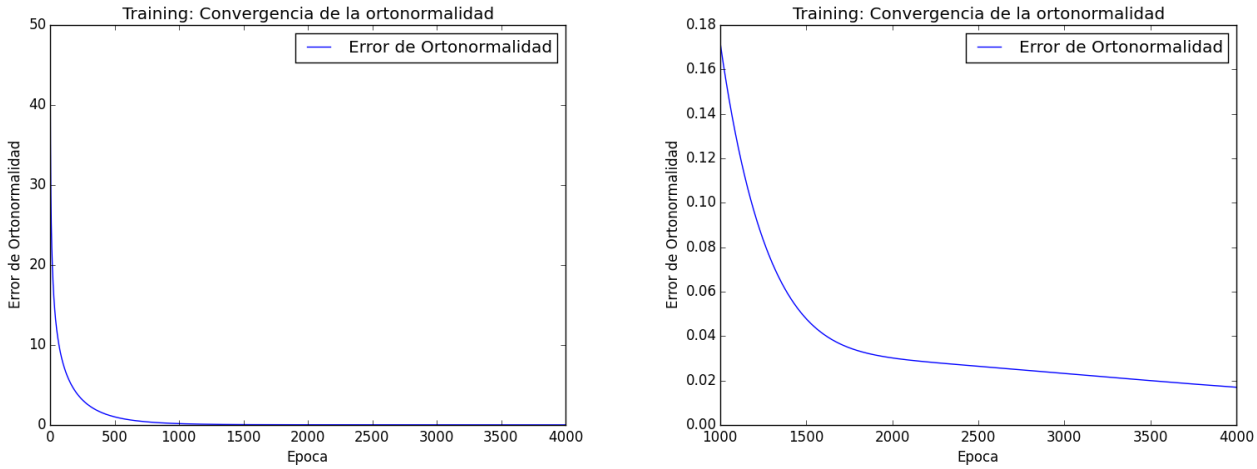


Figura 6: Varianza de los datos de entrada por categoría

En la figura 7, se puede observar como se satisface la condición de ortonormalidad, en función del número de época. Se aprecia que a partir de las 1000 épocas la norma se hace menor al 0.18 y continúa decreciendo fuértemente hasta valores por debajo de 0.02.



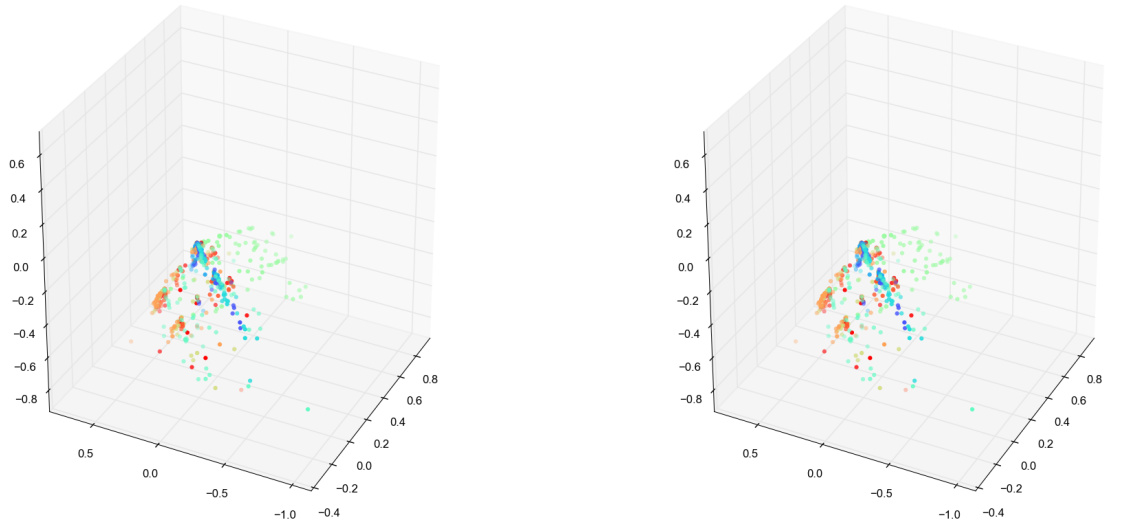
(a) Todas las épocas

(b) A partir de la época 1000

Figura 7: Ortonormalidad de la matriz de pesos en función de las épocas.

2.5.2. Algoritmo de Sanger

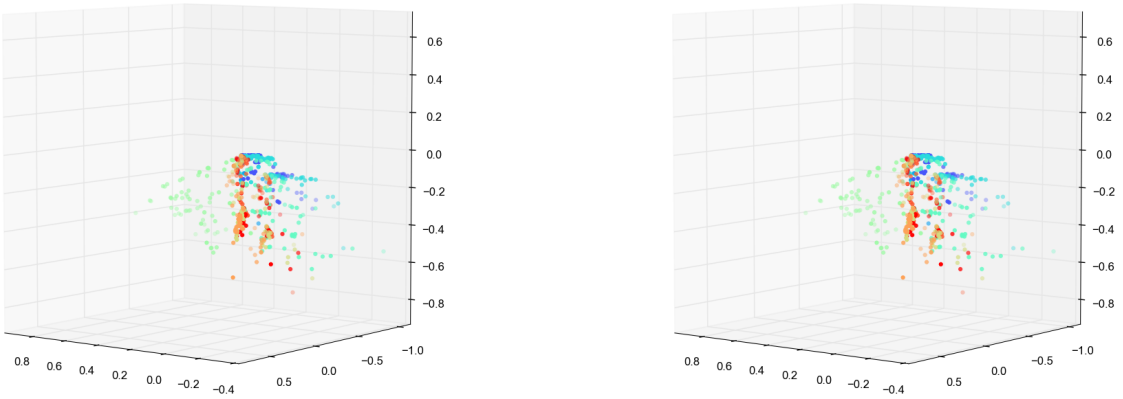
Se realizó una análisis similar al caso anterior, utilizando el mismo porcentaje de datos de entrenamiento - validación y el mismo coeficiente de aprendizaje adaptativo. En la figuras 8 y 9 se puede observar vistas 3d (componentes principales obtenidos por el algoritmo) de los datos de entrenamiento (figuras 8a, 9a) y validacion (figuras 8b, 9b). Al igual que en el caso anterior, obtenemos una buena clasificación sobre los datos de validación.



(a) Entrenamiento

(b) Validacion

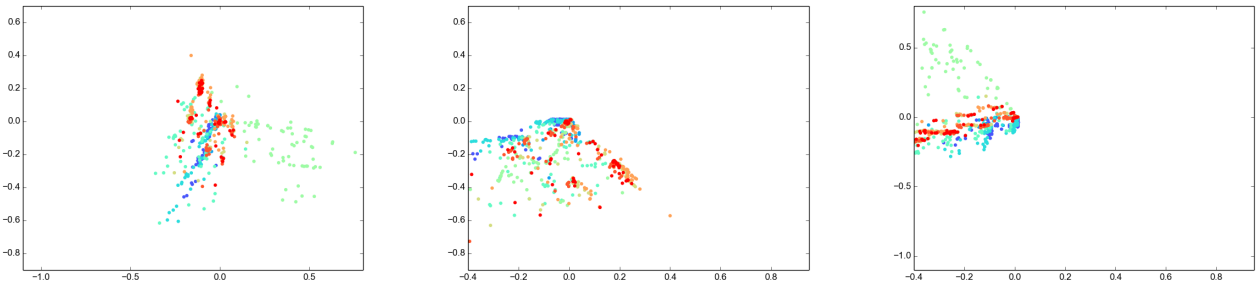
Figura 8: Vista 1 en 3d de resultados de validación y entrenamiento



(a) Entrenamiento (b) Validacion

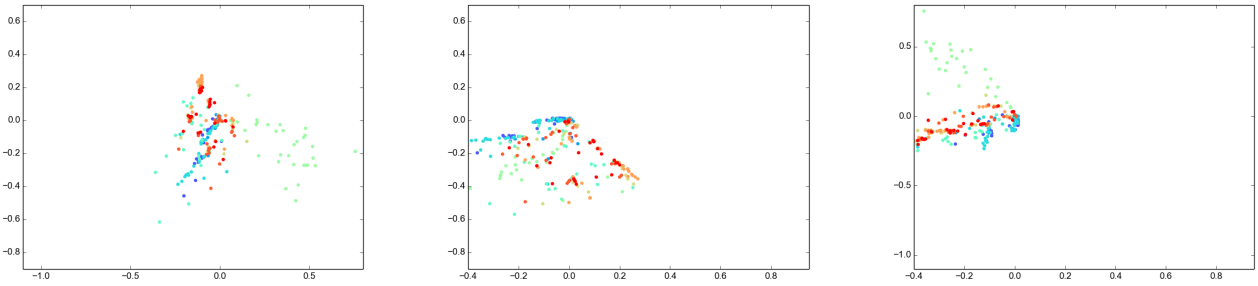
Figura 9: Vista 2 en 3d de resultados de validación y entrenamiento

proyectando estos gráficos 3d sobre distintos planos, obtenemos las siguientes figuras, tanto para entrenamiento figuras 10 como para validación figura 11



(a) Corte 1 (b) Corte 2 (c) Corte 3

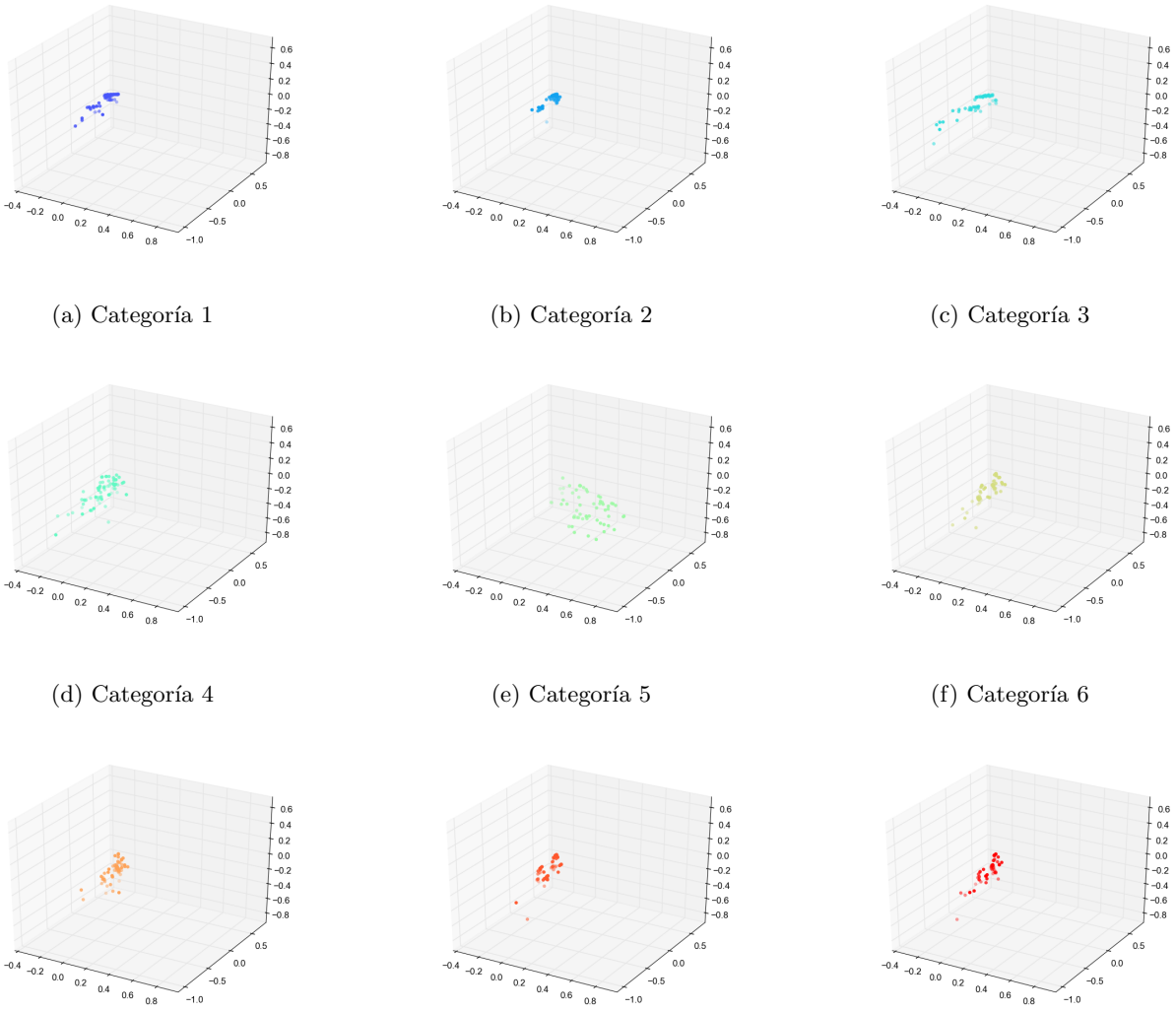
Figura 10: Cortes del gráfico de los datos de entrenamiento



(a) Corte 1 (b) Corte 2 (c) Corte 3

Figura 11: Cortes del gráfico de los datos de validación

A continuación podemos observar los resultados de entrenamiento en gráficos 3d para las distintas categorías, con el fin de observar más claramente como se distribuyen los mismos (figuras 12).



(a) Categoría 1 (b) Categoría 2 (c) Categoría 3

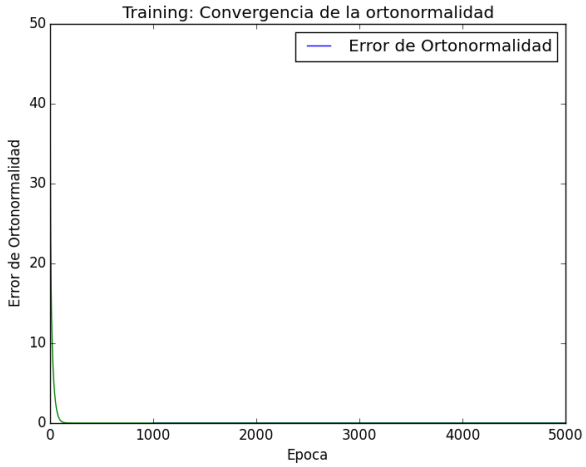
(d) Categoría 4 (e) Categoría 5 (f) Categoría 6

(g) Categoría 7 (h) Categoría 8 (i) Categoría 9

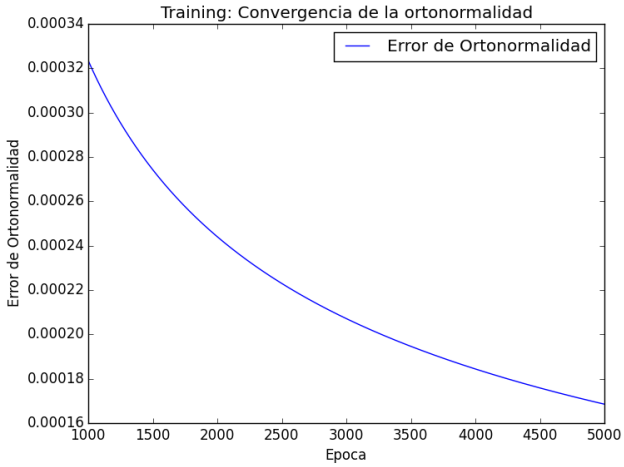
Figura 12: Gráficos de resultados diferenciados por categoría

Cabe destacar que los datos de entrada son los mismos tanto para Oja como para Sanger. Por lo que la varianza observada en la figura 6 también aplica para esta regla de aprendizaje y debemos tenerla en cuenta a la hora de comprender la dispersión de los datos de las categorías en los gráficos obtenidos.

En las figuras 13, se puede observar como se satisface la condición de ortonormalidad, en función del número de época. Se aprecia que a partir de las 1000 épocas la norma se hace menor al 0.0004, un valor muy bajo, y continúa decreciendo.



(a) Todas las épocas



(b) A partir de la época 1000

Figura 13: Ortonormalidad de la matriz de pesos en función de las épocas.

2.6. Conclusiones

Se implementaron dos algoritmos para reducir la dimensionalidad del set de datos de entrada. Tanto el algoritmo de Sanger como el de Oja nos permiten obtener una clasificación que es consistente con los datos de validación. La diferencia entre los algortimos radica en que el de sanger nos permite obtener las componente principales, mientras que el de oja nos da un subespacio generado por estos, pero no nos da explicitamente las direcciones. Contrario a lo esperado, no pudimos visualizar esta diferencia.

3. Mapeo de Características

Esta etapa del trabajo consiste en la implementación de un **mapa de características** para la clasificación de las palabras según la **categoría** que les fue asignada previamente.

Lo que se espera obtener es un **mapa auto-organizado** en el cual cada **categoría** este claramente diferenciada.

Para lograr el objetivo presentaremos un **mapa auto-organizado** basado en el algoritmo de *Kohonen*.

Detalleremos los pasos que realizamos para la implementación de la solución, la elección de la implementación y las dificultades que nos topamos al implementarlo.

3.1. Implementación del Algoritmo

Para implementar la solución tomamos el algoritmo de mapas de *Kohonen* visto en clase.

La versión que implementamos es la que realiza los calculos por **columna**, esta versión demora mas en los calculos que la versión matricial, pero su implementación nos resulto mas sencilla.

Tomamos una dimensíón del mapa de 20 x 20 y para la actualización de las vecindades utilizamos la **función gaussiana** con un **learning rate adaptativo**. Comenzando con un radio de vecindad de 10.

Para el entrenamiento utilizamos dos tipos de cota, una por **cantidad de epocas** y la otra por **la diferencia de la norma de la matriz de pesos entre dos epocas distintas**, cuando es menor a un delta se finaliza el entrenamiento.

3.2. Preprocesamiento de Datos

Analizando el **dataset** notamos que los datos son muy esparsos, hay gran cantidad de valores en 0 y muy pocos datos númericos.

Como primer medida corrimos el algoritmo con los datos sin procesar, luego los estandarizamos tomando varianza: 0 y sigma: 1. Pero no hubo mejoras en la clasificación.

El siguiente intento fue dividir los datos por 10, para llevarlo a valores entre 0 y 1, pero luego descartamos esta opción ya que no soportaría leves cambios en las escalas del **dataset**, al agregar algun parámetro de mayor magnitud.

Entonces decidimos dejar el **dataset** sin procesar y trabajar con los datos como vienen.

3.3. Dimensiones Adecuadas

Buscamos una dimensión adecuada para el mapa, conociendo que queremos clasificar 900 datos en 9 categorías.

Probamos con las siguientes dimensiones:

- 10 x 10
- 20 x 20
- 30 x 30

3.3.1. Dimensión 10 x 10

Obtuvimos que el mapa no era lo suficientemente grande y los valores de entrada se solapaban mucho, como lo muestra la figura:

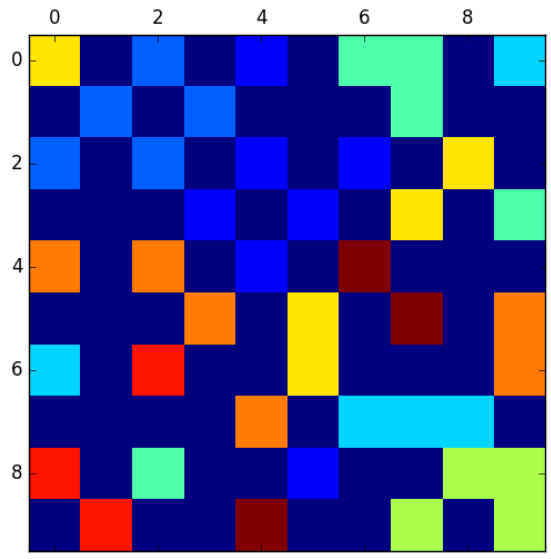


Figura 14: Mapa 10 x 10 para 200 entradas.

Por lo tanto descartamos esta configuración

3.3.2. Dimensión 20 x 20

Con estas dimensiones obtuvimos una buena distribución de las categorías, aunque hay un porcentaje de neuronas que no se activaron.

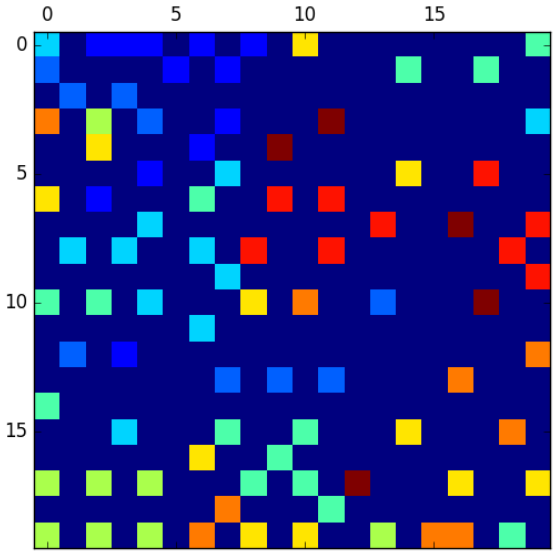


Figura 15: Mapa 20 x 20 para 200 entradas.

3.3.3. Dimensión 30 x 30

Para esta configuración obtuvimos mayor desperdicio de neuronas, es decir el porcentaje que nunca se activo fue muy alto, como lo muestra la siguiente figura

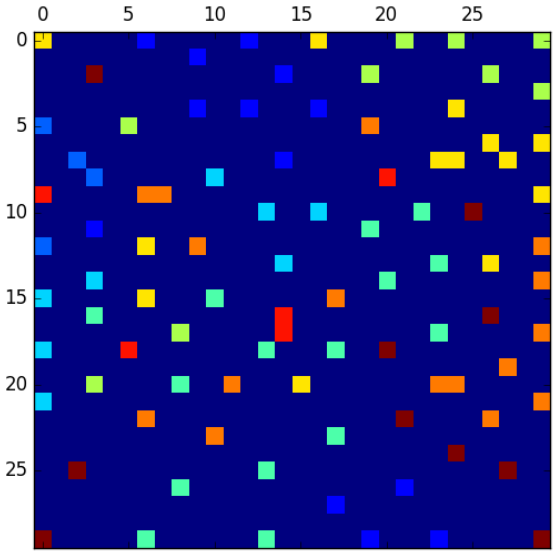


Figura 16: Mapa 30 x 30 para 200 entradas.

3.3.4. Conclusión sobre la Dimensiones

La que mejor se adapta a las características del problema, es la de dimensión de 20 x 20.

3.4. Vecindad, Learning Rate y Sigma

Para la elección de estos parametros consideramos las siguientes alternativas:
Vecindad por:

- Escalones: influenciamos vecinos a distancia n formando un cuadrado.
- Función Gaussiana: influenciamos n vecinos utilizando una función gaussiana.

Learning Rate por:

- Basado en la cantidad de epocas: definimos el learning rate basado en la cantidad de epocas aplicadas a una función.
- Coeficientes adaptativos: definimos un valor de learning rate inicial y lo disminuimos a traves de coeficientes.
- Decrecimiento porcentual: Definimos un valor de learning rate inicial y por epoca lo decrementamos en un 0.X por ciento.

Para el *Learning Rate* basados en la **Cantidad de Epocas**, probamos con las siguientes funciones: $\frac{1}{t^{\frac{-1}{2}}}$ y $\frac{1}{t^{\frac{-1}{3}}}$.

En el caso de **Coeficientes Adaptativos**, probamos con: learning rate inicial / (1 + epoca actual * coeficiente * learning rate inicial)

Y para del **Decrecimiento Porcentual**: por cada epoca learning rate inicial * 0.X

Sigma por:

Definimos un valor de *sigmainicial*, lo suficientemente grande para influenciar a todo el mapa. Y variandolo con las epocas con dos alternativas:

- $sigmaInicial * epocaActual^{\frac{-1}{3}}$
- $sigmaInicial * e^{\frac{epocaActual}{lambd}}$

Tomando como lambd: $cantidadEpocas * cantidadDatosEntrenamiento/log(sigmaInicial)$

3.4.1. Buscando la Combinación Ideal

Realizamos una bateria de prueba tomando un set de datos de 300 valores, para estudiar cual de la siguiente combinación de parametros organizaba mejor las categorías.

Primero definimos un sigma inicial: (dimensión mapa)/2, así en las primeras epocas gran parte del mapa seria influenciado.

De las primeras pruebas concluimos que el *Learning Rate* basado en la **cantidad de epocas** resultaba no ser una buena elección, ya que decrece de forma muy rápido sin dar el tiempo suficiente para aprender. Entonces descartamos esta opción.

Entonces utilizamos las otras dos opciones, definiendo un *Learning Rate* alto con valor de 0.999. En el caso de **Decrecimiento Porcentual** tuvimos el inconveniente de que o decrecía muy rápido o demasiado lento, haciendo dificil encontrar el punto justo.

El que nos permitió encontrar el balance fue **Coeficientes Adaptativos** tomando como coeficiente = 0.5.

La función de **Sigma** basada en $sigmaInicial * e^{\frac{epocaActual}{lambd}}$ nos dió que decrecia de forma muy rápido, impidiendo el aprendizaje. La función $sigmaInicial * epocaActual^{\frac{-1}{3}}$ *combinadoal Learning Rate* de tipo **adaptativo** nos dió un buen balance.

Para la función de vecindad decidimos quedarnos con la **función uniforme** ya que presentaba una distribución más ármonica.

3.4.2. Conclusiones de los Parámetros

La combinación que nos resulto mejor fue tomar:

- Sigma Inicial = 10
- Learning Rate Adaptativo = $\frac{0{,}999}{1+epoca\_actual \cdot 0{,}5 \cdot 0{,}999}$
- Sigma = $SigmaInicial \cdot epocaActual^{\frac{-1}{3}}$

3.5. Cotas

Para finalizar el entrenamiento definimos dos tipos de cotas:

- Epocas
- Norma

La basada en la cantidad de **Epocas** establece una cantidad máxima de ciclos de entrenamiento.

La basada en la **Norma** calcula la norma de la matriz de pesos (W) correspondiete a la epoca actual, hace la diferencia con la epoca anterior. En caso que sea menor a una epsilon termina el entrenamiento. La ventaja de esta es que cuando los pesos se modifican de forma despreciable entre dos epocas significa que el aprendizaje comienza a estancarse.

3.6. Entrenamiento

Con los parametros seleccionados procedemos a realizar el entrenamiento de nuestro mapa, para esto nos queda determinar la cantidad de datos con los que conviene entrenar y estudiar si hay alguna diferencia variando las entradas y las cotas con las que se finaliza el entrenamiento.

Tomamos distintos valores de entrada para entrenar, realizamos los test, y reentrenamos el mapa nuevamente durante 3 etapas.

Con estas pruebas esperamos ver con que cantidad de datos de entrenamiento, con cual de las cotas y si la repetición influyen en la calidad de los resultados.

Al ingresar el parametro de entrenamiento, se toma una porción del dataset correspondiente a la cantidad de entradas de forma random.

Los resultados del entrenamiento, los clasificamos en 3 categorías **Bien**, **Mal** y **Sin Determinar** esta última corresponde a que existen dos o mas categorias a la que podría pertenecer.

3.6.1. 50 Datos de Entrada

Primero realizamos una prueba para ver si tomando un pequeña cantidad de datos alcanza para categorizar todos los valores de una forma eficiente.

Configuramos los siguientes parametros como cotas:

- Cantidad Máxima de Epocas = 1000
- Cota de la norma = 0.00001
- Learning Rate = 0.999

Y ejecutamos la función de entrenamiento.

Al finalizar consumió las 1000 epocas, y manteniendo una diferencia den la norma del orden 0.008 y disminuyendo, eso quiere decir que si bien gran parte se había agrupado, todavía podía seguir corrigiendose.

Guardamos el mapa de entrenamiento y repetimos el proceso para el mismo mapa dos veces mas, para analizar si el entrenamiento en etapas produce mejoras.

En las dos etapas siguientes nuevamente volvio a consumir las 1000 epocas de entrenamiento, sin alcanzar la cota de la norma.

En cada etapa ejecutamos la función **Test** y volcamos los resultados obtenidos de la clasificación en una tabla:

Como se puede ver en entrenamiento mejoró levemente respecto a la primer etapa, para la tercera se generó una disminución respecto a los aciertos, pero una mejor clasificación de los incorrectos.

Presentamos el mapa obtenido del entramiento para la tercera etapa.

Etapa	Bien	Mal	Sin Determinar
1	465	367	17
2	492	285	72
3	414	400	35

Cuadro 1: Resultados de Validación

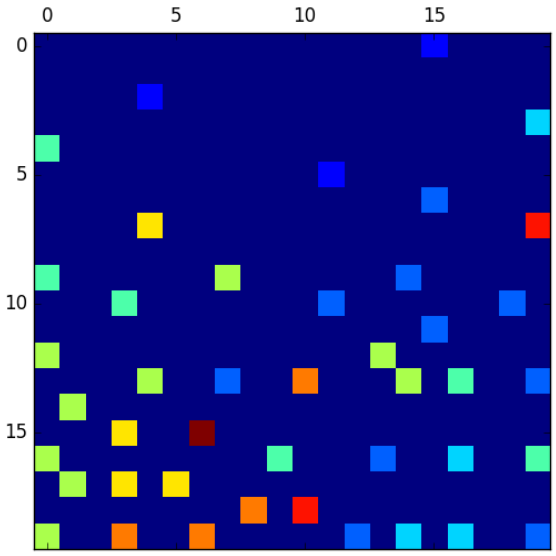


Figura 17: Mapa de Entrenamiento para 50 entradas.

Y para contrastar presentamos el mapa obtenido de la validación

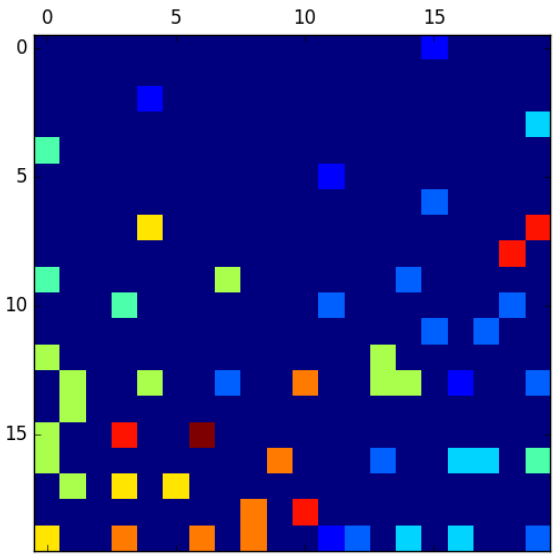


Figura 18: Mapa de Entrenamiento para 50 entradas.

Para detectar las áreas que fueron mejor clasificadas, graficamos el mapa de los aciertos.

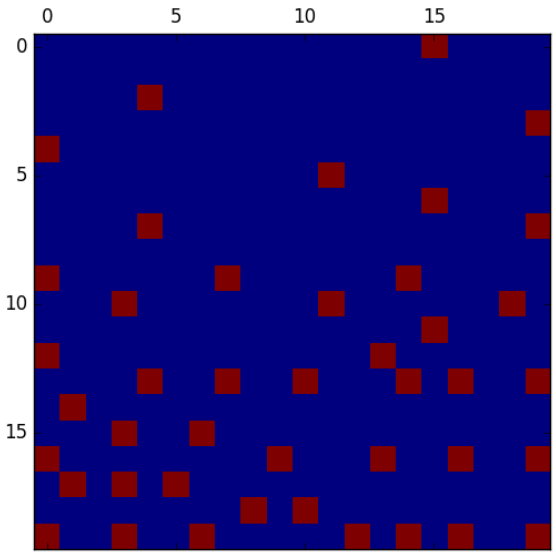


Figura 19: Mapa de Aciertos para 50 entradas.

Hacemos lo mismo con los datos mal clasificados.

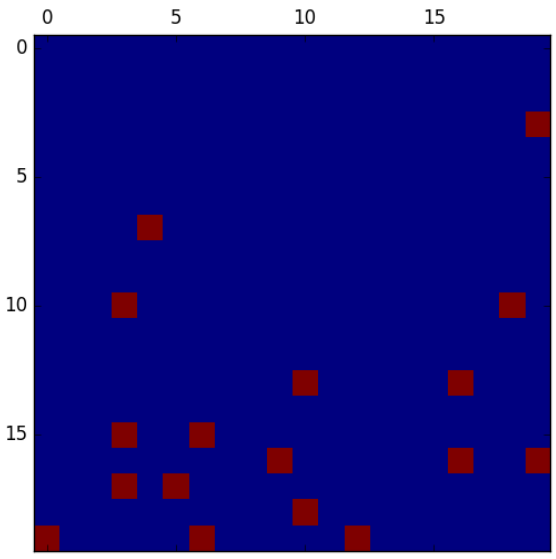


Figura 20: Mapa de Errores para 50 entradas.

Y para los datos sin determinar.

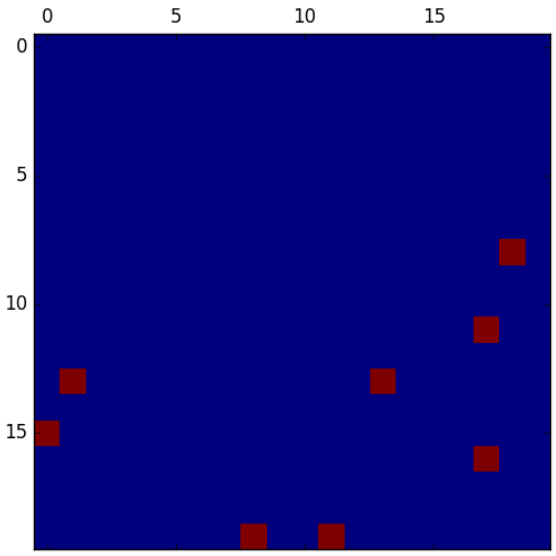


Figura 21: Mapa de Indefinidos para 50 entradas.

Como se puede ver la mayor cantidad de malas clasificaciones y e imposibilidad de clasificar se da en el área donde las neuronas no llegaron a organizarse completamente.

3.6.2. 200 Datos de Entrada

Incrementamos los datos de entrenamiento, con la premisa de que un mayor set de entrenamiento va a mejorar la clasificación.

Asique configuramos el algoritmo con los siguientes parametros para 200 datos

- Cantidad Máxima de Epocas = 300
- Cota de la norma = 0.00001
- Learning Rate = 0.999

Disminuimos la cantidad de épocas, ya que al ser mas datos, pensamos que se organizarían mas rápido.

Corrimos el entrenamiento y nos consumió las 300 épocas. Guardamos el mapa y repetimos para una siguiente etapa con la configuración

- Cantidad Máxima de Epocas = 500
- Cota de la norma = 0.00001
- Learning Rate = 0.999

Nuevamente volvió a consumir la cantidad total de epocas. Entonces para la etapa 3 lo configuramos con los siguientes parámetros:

- Cantidad Máxima de Epocas = 1000
- Cota de la norma = 0.00001
- Learning Rate = 0.999

Terminada la tercer etapa, también consumió las 1000 epocas.

Para cada etapa de entrenamiento realizamos la validación y volcamos los resultados en la siguiente tabla:

Etapa	Bien	Mal	Sin Determinar
1	481	201	17
2	494	192	13
3	470	198	31

Cuadro 2: Resultados de Validación

En este caso tuvimos una mayor tasa de aciertos, y el entrenamiento resulto en mejoras progresivas.

Además el mapa generado del entrenamiento logra un mejor agrupamiento de las categorías

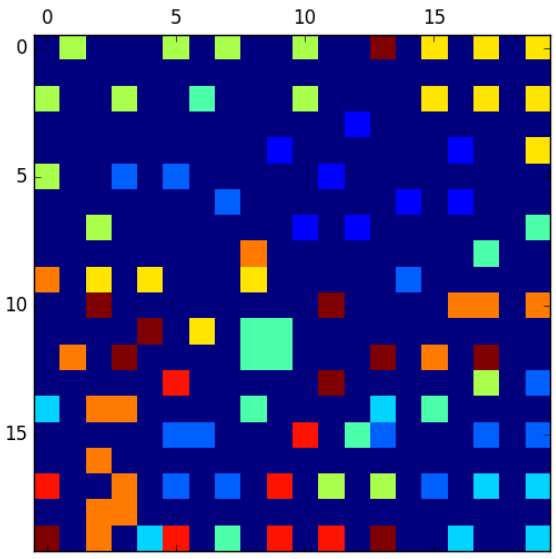


Figura 22: Mapa de Entrenamiento para 200 entradas.

Lo comparamos con el mapa obtenido de la validación y vemos que hay una mejor correlación, respesto a la primer prueba con menos datos

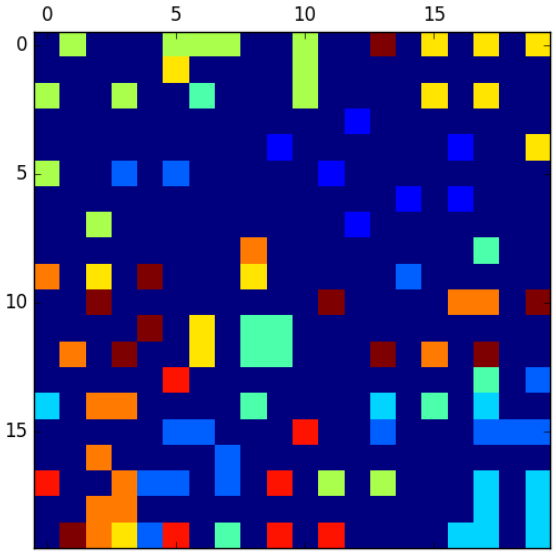


Figura 23: Mapa de Entrenamiento para 200 entradas.

Presentamos el mapa de los aciertos:

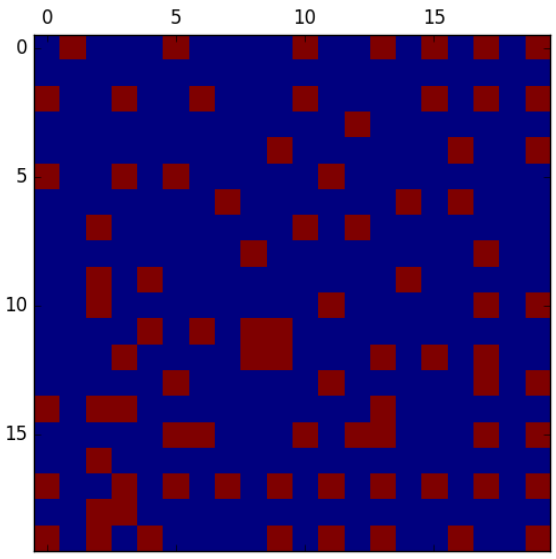


Figura 24: Mapa de Aciertos para 200 entradas.

Se pudo lograr buen porcentaje de aciertos para muchas de las categorías.
Presentamos el mapa de las mal clasificadas:

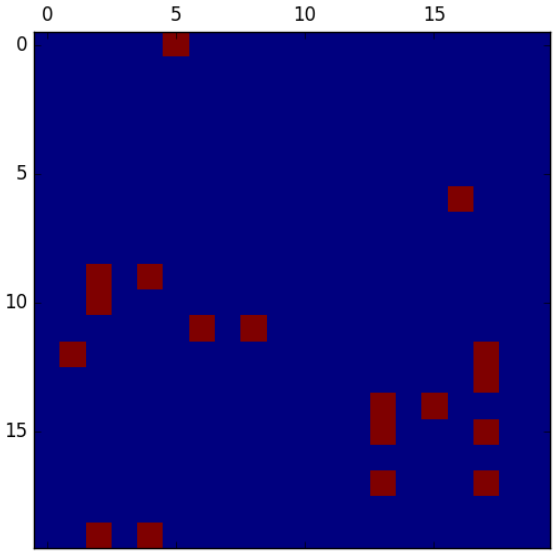


Figura 25: Mapa de Errores para 200 entradas.

Vemos que la mayor parte de los errores se centra de nuevo en el area de mayor error de categorización
Y vemos los que no pudieron clasificarse:

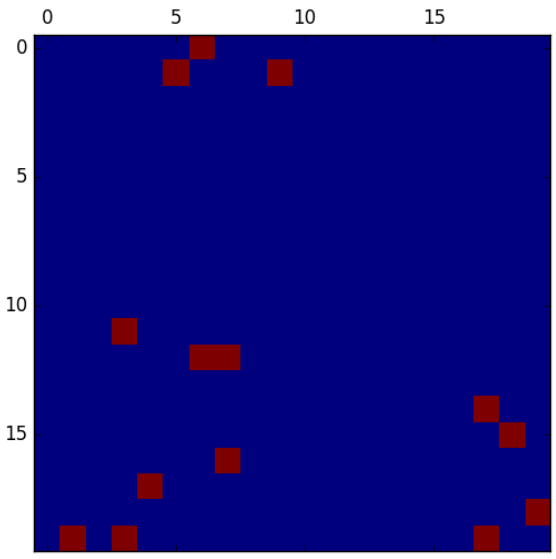


Figura 26: Mapa de Indefinidas para 200 entradas.

Aquí no pudimos establecer un patrón.

Por lo que notamos a mayor cantidad de datos tomados del dataset de entrenamiento mejor resulta la clasificación. Por eso para el siguiente caso vamos a tomar un dataset mucho mayor.

3.6.3. 800 Datos de Entrada

Por lo analizado hasta ahora, cuanto mas datos tengamos en entrenamiento, mejor sería la clasificación, para comprobar esto tomamos un set de entrenamiento de 800 entradas.

Configuramos el algoritmo con los siguientes parametros:

- Cantidad Máxima de Epocas = 10000
- Cota de la norma = 0.00001
- Learning Rate = 0.999

Asignamos una cantidad de epocas mas grande que en los anteriores casos, ya que no teniamos idea de cuanto podria tardar en organizarse.

Ejecutamos el entrenamiento y en solo 368 epocas alcanzó la cota de la norma.

Repetimos el mismo entreamiento y en la siguiente etapa alcanzó la cota de la norma en la época 687. En la tercer etapa fue en la 645.

Para cada etapa nuevamente ejecutamos la función test, para validar los datos y presentamos los resultados obtenidos en la siguiente tabla:

Logramos una mayor tasa de aciertos, insumiendo menos epocas de entrenamiento que en los casos anteriores.

Presentamos el mapa de entrenamiento obtenido para la tercer etapa:

Etapa	Bien	Mal	Sin Determinar
1	74	25	0
2	83	16	0
3	77	22	0

Cuadro 3: Resultados de Validación

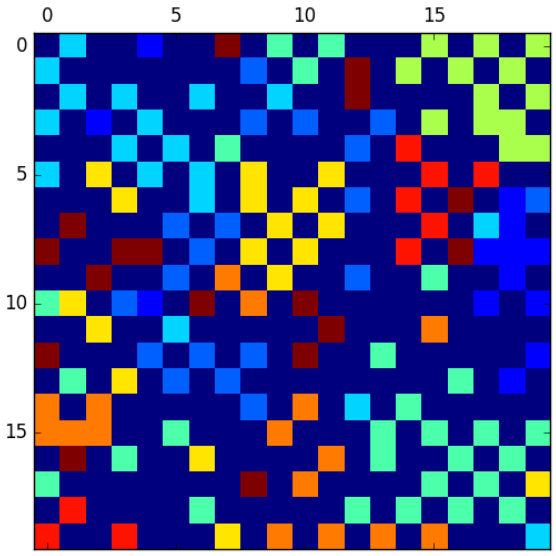


Figura 27: Mapa de Entrenamiento para 800 entradas.

Y lo contrastamos contra el mapa de validación

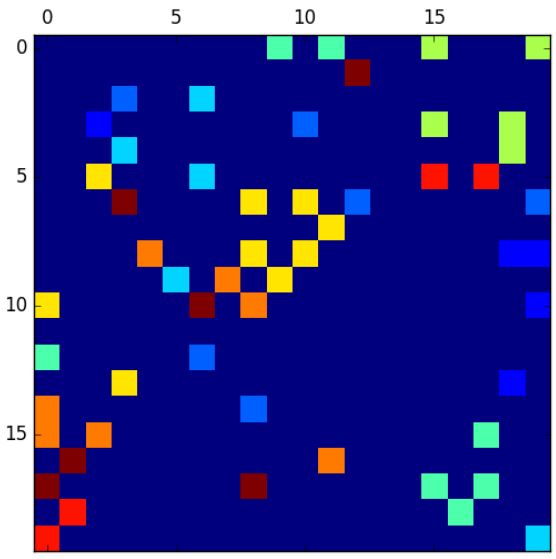


Figura 28: Mapa de Entrenamiento para 800 entradas.

En este caso podemos ver una mejor clasificación de las categorías
Presentamos el mapa de los aciertos:

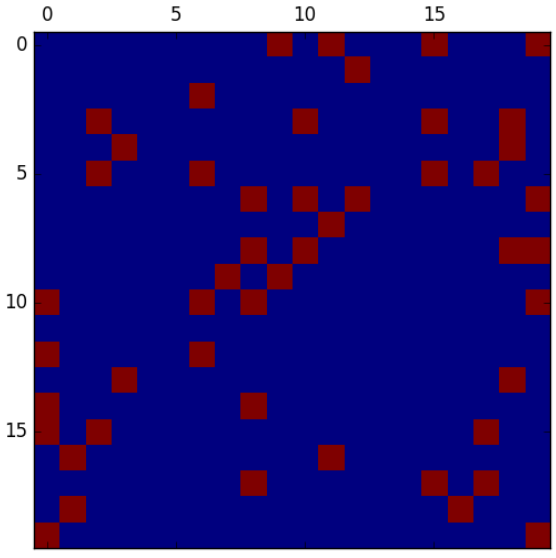


Figura 29: Mapa de Aciertos para 800 entradas.

Notamos que las bien categorizadas cubren casi todo el mapa.
Ahora veamos que áreas presentaron los mayores problemas y se clasificaron mal

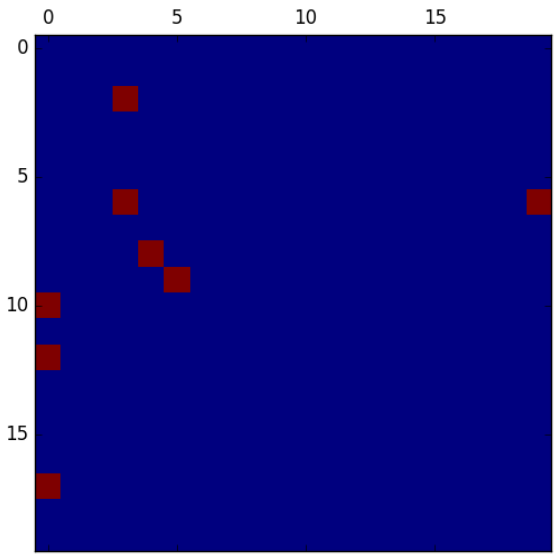


Figura 30: Mapa de Errores para 800 entradas.

Podemos ver que los errores se concentran en las areas bordes que del mapa de entrenamiento se ve que no quedo del todo separado.

3.6.4. El mejor mapa

De los casos analizados, el mapa que nos dio los mejores resultados corresponden al de 800 entradas de entrenamiento, en la etapa 2.

Mostramos aquí como nos queda el mapa:

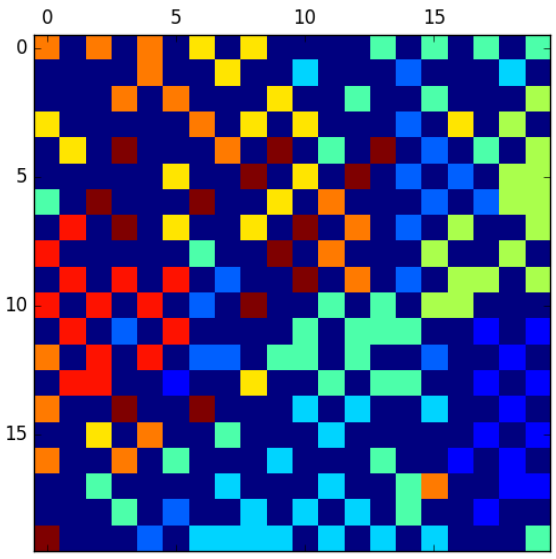


Figura 31: Mapa de Entrenamiento.

Lo contrastamos contra el mapa generado por la validación

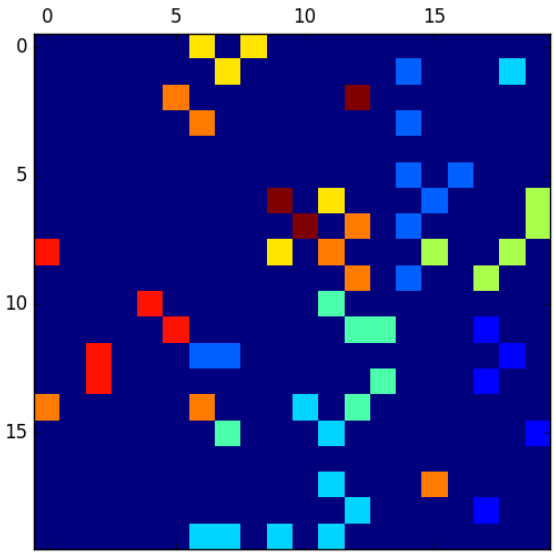


Figura 32: Mapa de Entrenamiento.

Mapa de Aciertos:

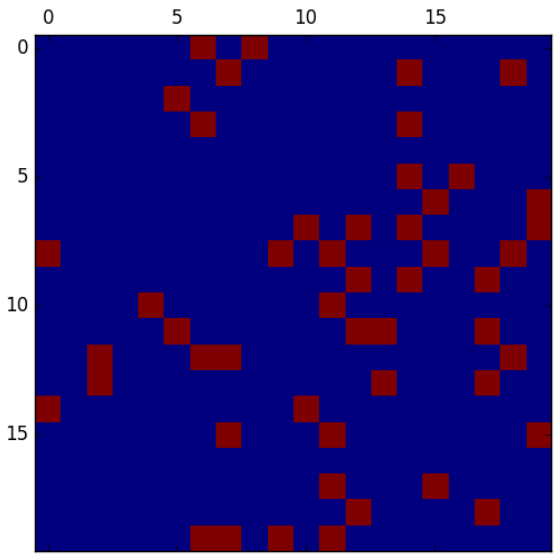


Figura 33: Mapa de Aciertos.

Mapa de Errores:

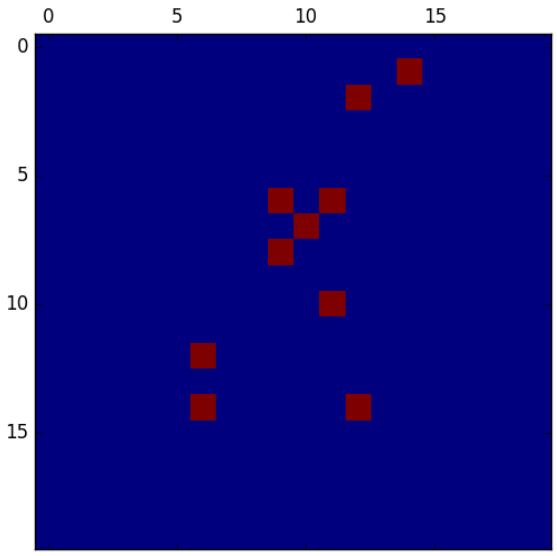


Figura 34: Mapa de Errores para 800 entradas.

3.6.5. Futuros Entrenamientos

Luego de analizar los casos vemos que agregando etapas de entrenamiento, la clasificación mejora. Nos queda pendiente seguir entrenando los mapas hasta encontrar el punto de balance donde no se produzcan mejoras significativas entre etapas de entrenamiento.

3.7. Detalles de Implementación

La implementación de la solución fue desarrollada usando el lenguaje python, con la librería numpy para facilitar las operaciones aritméticas y los gráficos fueron construidos con la librería matplotlib.pyplot.

Para armar el mapa de Kohonen implementamos una clase **Kohonen** con la siguiente estructura:

```
struct kohonen {
  learning_rate = coeficiente de aprendizaje.
  tolerancia_error = cota de salida del entrenamiento para la diferencia entre normas.
  cantidad_epocas = cantidad máxima de epocas de entrenamiento.
  dimension = dimensión del mapa.
  entradas = cantidad de archivos usados para entrenar.
  input_file = archivo de entrada del dataset.
  data_entrenamiento = inicializado en 0.
  data_validacion = inicializado en 0.
  N = fijo en 856
  M1 = M2 = dimensión de entrada
  M = M1* M2
  W = matriz de pesos inicializada en random.
  cant_categorias = fijo en 9
  Mres = resultado de las categorias, inicializado en 0
  actualizarDataSet() = toma valores de entrada random segun la cantidad de entradas
}
```

Para ejecutar el algoritmo correr: python ejercicio2.py

Para acceder al menú de ayuda ingresar: help

Para iniciar un entrenamiento ingresar: train

Para validar los datos ingresar: test

Para guardar los resultados de entrenamiento ingresar: export nombreMapa.in

Para importar un mapa ingresar import nombreMapa.in

Para salir del programa ingresar: exit

3.8. Detalles de Resultados

Los resultados correspondientes a la sección de entrenamiento son adjuntados con la siguiente estructura:

XXX Entradas/Etapa X/: la carpeta Entradas corresponde a la cantidad de datos usados para entrenar la red y la subcarpeta Etapa corresponde a una etapa de entrenamiento. Dentro de la carpeta de Etapas se encuentran los siguientes archivos

- mapa XXX X.in = corresponde al mapa entrenado.
- resultados mapa XXX X.txt = corresponde a los resultados de ejecutar la validación sobre el entrenamiento.
- mapa XXX X aciertos.png = muestra la posición en la mapa de los registros bien clasicados.
- mapa XXX X errores.png = muestra la posición en la mapa de los registros mal clasicados.
- mapa XXX X indefinidas.png = = muestra la posición en la mapa de los registros que no se han podido clasificar.

- mapa XXX X entrenamiento = es el mapa obtenido del entrenamiento.

3.9. Conclusiones

En nuestro experimento, la implementación propuesta del mapa auto organizado basado en el algoritmo de **mapa de Kohonen** alcanzo una efectividad del 80 por ciento de aciertos en nuestra mejor ejecución. Estamos convencidos de que realizando mas etapas de entrenamiento podemos mejorar ese número.

Como conclusión notamos que la cota basada en la norma, es mas efectiva que la de cantidad de epocas, ya que nos da una medida justa de cuanto esta modificandose la matriz de pesos, es decir, sobre cuanto esta aprendiendo con el correr de las epocas.