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.

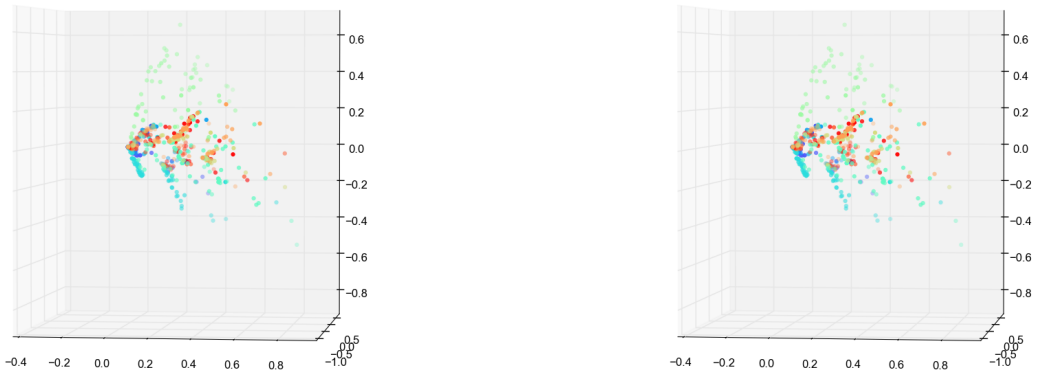
2.4. Experimentación y Resultados

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(Wt*W - 1 = 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 1/(epoca)* alfa, con alfa entre 0 y 1. Probando estas distintas opciones encontramos que si bien no existían diferencias significativas, un coeficiente del tipo 1/(epoca) *1/2 presentaba los mejores resultados.

2.4.1. Algoritmo de Oja

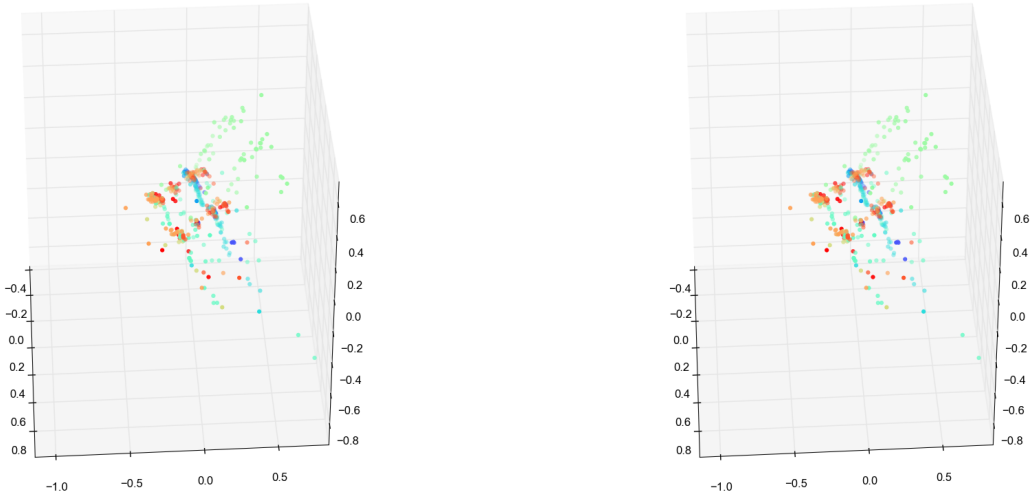
A continuación se pueden observar los resultados obtenidos utilizando el algoritmo de Oja separando el set de datos de entrada en 70 % - 30 %, para el coeficiente de aprendizaje adaptativo propuesto. La figura 1a presenta los datos en el espacio 3d de salida obtenido por el algoritmo. La figura 1a corresponde a los datos de entrenamiento, mientras que la figura 1b corresponde a datos de validación.

Se puede observar que la validación es consistente con la clasificación obtenida en la etapa de entrenamiento, dando cuenta de la validez del proceso. También se puede observar claramente en el gráfico la separación entre las



(a) Entrenamiento (b) Validacion

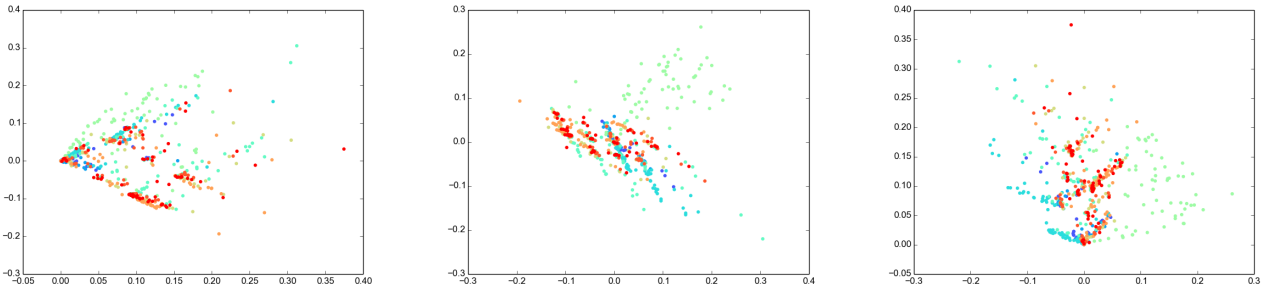
Figura 1: texto de abajo



(a) Entrenamiento (b) Validacion

Figura 2: texto de abajo

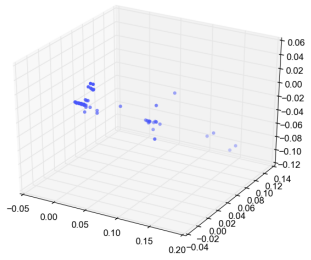
clases verde y azul, no siendo así para las clases restantes. Para poder discriminar entre estas últimas realizamos graficos 2d (figura 1.2), proyectando el gráfico 3d sobre distintos ejes.



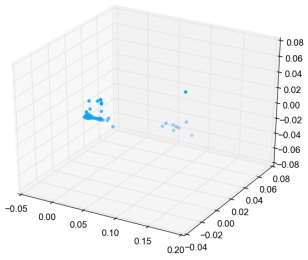
(a) Corte 1 (b) Corte 2 (c) Corte 3

Se puede observar...

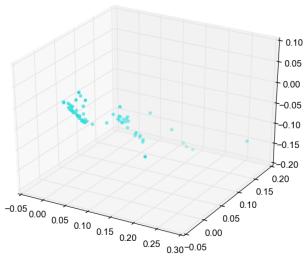
Por último realizamos un gráfico 3D para cada clase en la tapa de entrenamiento, con la finalidad de po. Los resultados se observan en la figura 1.3



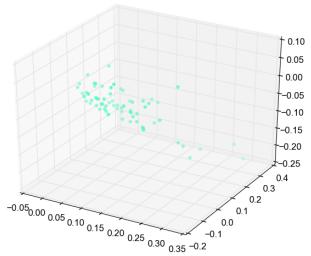
(a) Categoría 1



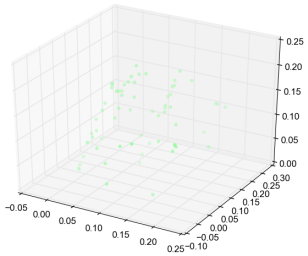
(b) Categoría 2



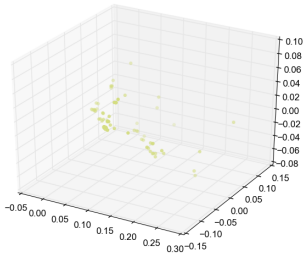
(c) Categoría 3



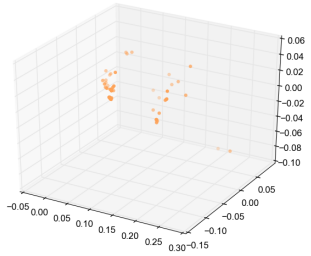
(d) Categoría 4



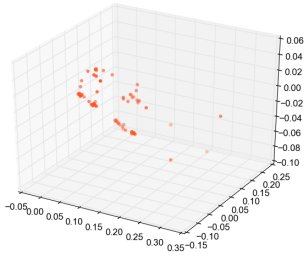
(e) Categoría 5



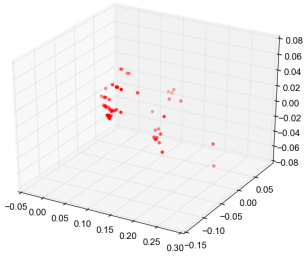
(f) Categoría 6



(g) Categoría 7



(h) Categoría 8



(i) Categoría 9

2.4.2. Algoritmo de Sanger

2.5. Conclusiones

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.

Dividimos el informe en las siguientes secciones:

- Implementación del Algoritmo: detallamos de forma introductoria la implementación del algoritmo. En las siguientes etapas discutiremos los pasos hacia la implementación final.
- Dimensiones Adecuadas: discutimos las alternativas para la dimensión del mapa.
- Vecindad, Learning Rate y Sigma: discutimos las alternativas que estudiamos para seleccionar los parámetros.
-

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. 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

3.2.1. 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:

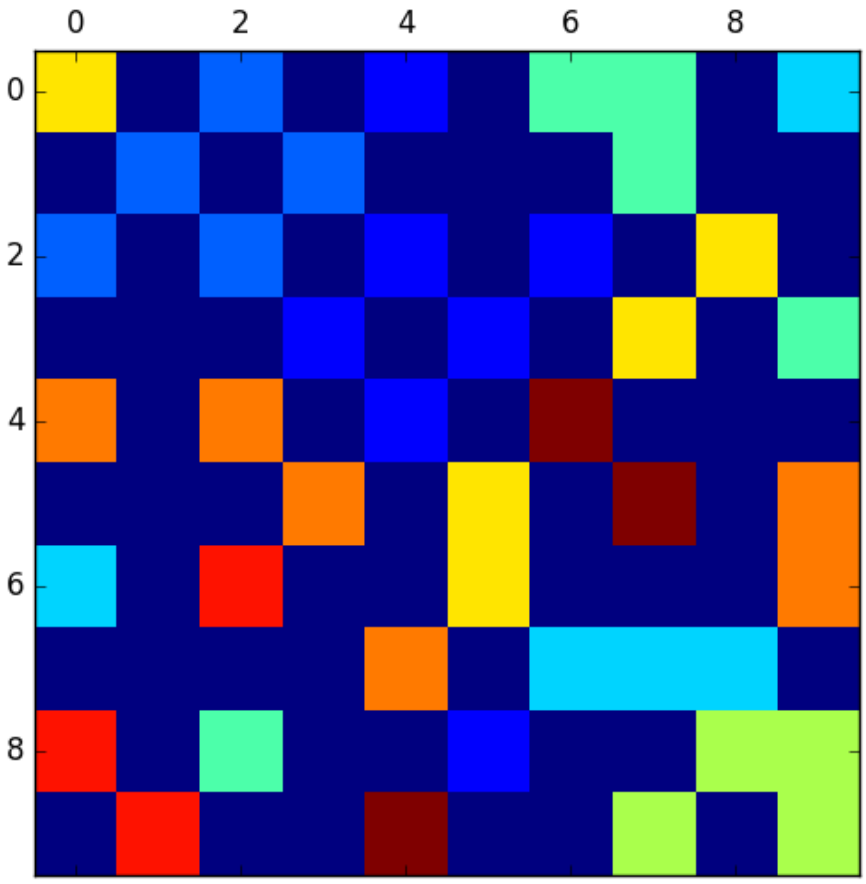


Figura 5: Mapa 10 x 10 para 200 entradas.

Por lo tanto descartamos esta configuración

3.2.2. Dimensión 20 x 20

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

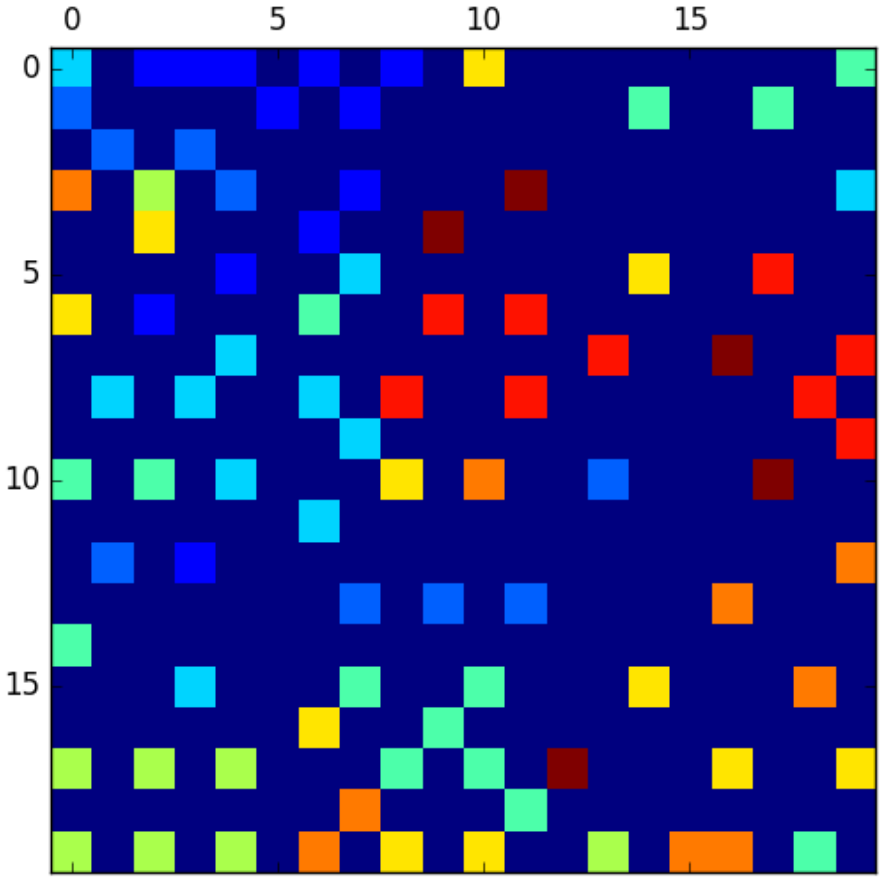


Figura 6: Mapa 20 x 20 para 200 entradas.

3.2.3. 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

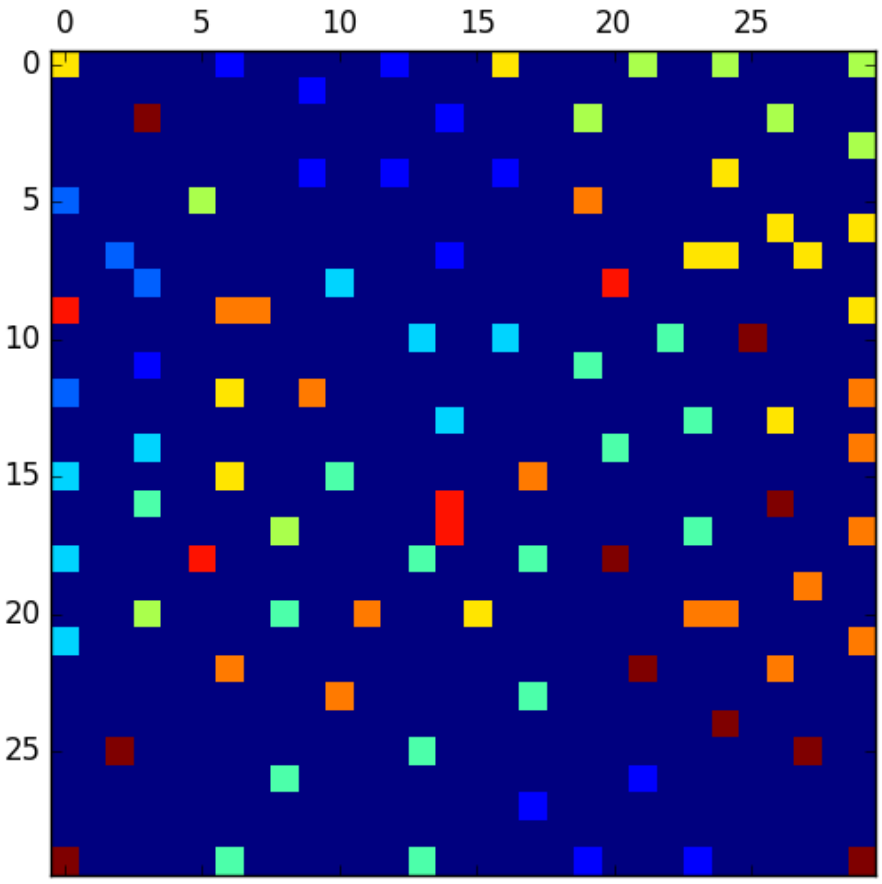


Figura 7: Mapa 30 x 30 para 200 entradas.

3.2.4. Conclusión sobre la Dimensiones

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

3.3. 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:

- $\frac{1}{t^{\frac{-1}{2}}}$
- $\frac{1}{t^{\frac{-1}{3}}}$

En el caso de **Coeficientes Adaptativos**, probamos con: $learningrateinicial/(1 + epoca_actual * coeficiente * learningrateinicial)$

Y para del **Decrecimiento Porcentual**: por cada epoca $learningrateinicial * 0.X$

Sigma por:

Definimos un valor de $sigma_inicial$, 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: $cantidadEpocas * cantidadDatosEntrenamiento/log(sigmaInicial)$

3.3.1. 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 $sigmaInicial * epocaActual^{\frac{-1}{3}}$ combinado al *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.

3.3.2. Conclusiones de los Parámetros

La combinación que nos resulto mejor fue tomar:

- $SigmaInicial = 10$
- $LearningRateAdaptativo = 0{,}999/(1 + epoca_actual * 0{,}5 * 0{,}99)$
- $Sigma = SigmaInicial * (epocaActual^{\frac{-1}{3}})$

3.4. 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.

3.5. 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.

3.5.1. 50 Datos de Entrada

3.5.2. 200 Datos de Entrada

3.5.3. 800 Datos de Entrada

3.6. 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

3.7. 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.

3.8. Conclusiones