DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Redes Neuronales

Trabajo práctico 2

Resumen

Entrenamiento de Redes Neuronales

Integrante	LU	Correo electrónico
Negri, Franco	893/13	franconegri2004@hotmail.com
Podavini Rey, Martín Gastón	483/12	marto.rey2006@gmail.com

Palabras claves:

TP

Índice

1. Introduccion

En este trabajo, nos disponemos a utlizar diferentes tecnicas de aprendizaje no supervisado para clasificar textos con ciertas caracteristicas. Los mismos consisten en descripciones de diversas companías y nuestro objetivo será lograr clasificar cada una en una categoria correspondiente. Contamos, ademas, con las verdaderas categorias de cada compañía para realizar una validación de los datos. Comenzaremos con una breve descripcion de los algoritmos para entender cual es el proposito y ejemplificaremos con casos de la vida real.

PCA: Un problema comun en reconocimiento de patrones es la de seleccionar atributos. Es decir, tenemos un espacio de datos (de informacion sobre algun fenomeno que queremos estudiar, por ejemplo reconocimiento de mails de SPAM) y lo transformamos en un espacio de atributos, es decir por ejemplo, tenemos un dataset de mails y nos quedamos con la cantidad de aparicion de ciertas palabras solamente como atributos. Sin embargo, esta transformacion a veces se hace de manera que el data set pueda ser representado por un reducido numero de atributos .efectivos"(es decir, atributos que me aporten mucha informacion sobre el tipo de mail que es). Para ser mas especificos, supongamos que tenemos un vector x de m dimensiones, y queremos transmitirlo usando l numeros, con l ¡m, lo que implica que estamos comprimiendo datos. Si simplemente truncamos el vector, causamos un error equivalente a la suma de las varianzas de los elementos eliminados de x, entonces lo que nos preguntamos es: Existe una transformacion lineal T tal que el truncamiento de Tx tal que su error cuadratico medio sea minimo? A dicha transformacion es la que llamamos obtener las componentes principales de un vector. Veamos un ejemplo:

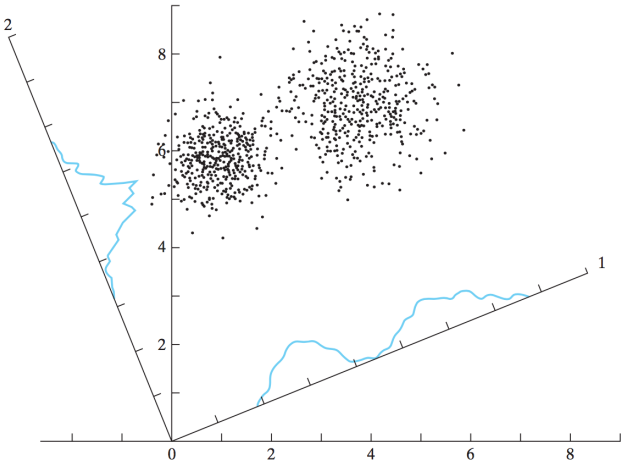


FIGURE 8.4 A cloud of data points is shown in two dimensions, and the density plots formed by projecting this cloud onto each of two axes, 1 and 2, are indicated. The projection onto axis 1 has maximum variance and clearly shows the bimodal, or clustered, character of the data.

SOM: Es un tipo de aprendizaje no supervisado de tipo competitivo. Que queremos decir con esto? Tenemos un conjunto de neuronas de salida organizadas en una grilla de 2 dimensiones (tambien conocido como "lattice"), dichas neuronas çompiten.entre ellas para ser activadas. Pero solo una de ellas sera la que disparará (también podemos generalizarlo a grupos de neuronas, y que disparara solamente una neurona de cada grupo). Las neuronas que se activan (aquellas que ganan la competencia) las llamaremos neuronas ganadoras. Bajo esta visión lo que nosotros haremos es conocido en la literatura como winner-takes-all. Estas neuronas son inducidas a distintos estimulos (o patrones de distintas clases) para que se establezca un orden entre neuronas ganadoras. Es decir, lo que trataremos de hacer es que bajo patrones similares, vamos a querer que se activen neuronas que sean cercanas entre ellas bajo alguna metrica. En nuestra grilla podemos tomar como metrica la distancia euclediana y lo que vamos a querer respetar es que bajo un conjunto de inputs que sean muy distintos, activaciones de neuronas que esten alejadas, produciendo en nuestro

mapa de neuronas çlusters.º grupos bien diferenciados de neuronas en estado exitatorio. Es decir, preservamos una topologia, a entradas de distintas clases le corresponden neuronas de distintos grupos, y cuanto mas disimilares sean estas entradas, mas lejanas seran las neuronas activadas. La motivacion de esto fue que por ejemplo, el sistema tactil (Kaas et cl., 1983), visual (Hubel and Wiesel, 1962, 1977) y acustico (Suga, 1985) son "mapeados.ª diferentes areas en la corteza cerebral respetando un orden topologico. Los mapas computacionales ofrecen cuatro propiedades importantes (Knudsen et a., 1987; Durbin and Michison, 1990).

1. En cada mapa, la neuronas actuan en paralelo y procesan pedazos de informacion que son similares en su naturaleza, pero proceden de diferentes reginoes en el espacio de entrada sensorial.

2. En cada etapa de la representación, cada pieza de entrada de información se mantiene en su contexto.

3. Las neuronas que tratan con inputs cercanos (parecidos) son cercanas entre ellas por lo que interactuan a travez de conexiones sinapticas cortas.

4. Los mapas contextuales pueden ser entendidos en terminos de mapas de reduccion de una dimension alta.

Veamos una posible implementación de nuestra grilla dado un input:

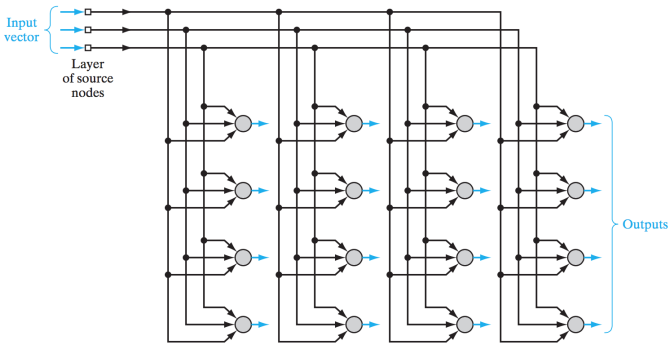
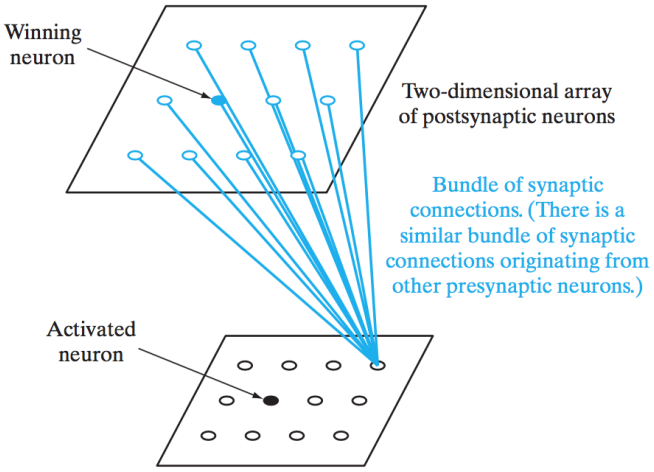
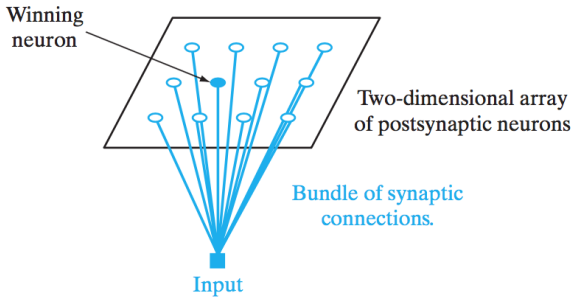


FIGURE 9.2 Two-dimensional lattice of neurons, illustrated for a three-dimensional input and four-by-four dimensional output (all shown in red).

Hay dos modelos implementativos para este tipo de mapas. Uno es el de Willshaw-von der Malsburg y el segundo el de Kohonen. En particular nos especializaremos en el segundo pero vamos a dar una breve reseña de cada uno.



(a) Willshaw–von der Malsburg's model



(b) Kohonen model

El primero lo que hace es que tanto la entrada como la salida tienen la misma dimension. Es decir que si estamos usando un mapa de 2 dimensiones como salida, vamos a tratar la entrada como una grilla de neuronas de 2 dimensiones. Para entenderlo mejor, a continuacion un grafico.

Algo a remarcar es que en este modelo las neuronas postsinapticas no son del tipo winner-takes-all sino que se utiliza un threshold para asegurar que solo unas pocas son activadas. Ademas, cada uno de los pesos tiene una cota superior establecida para que no se supere y lleve a inestabilidades de la red. Su motivacion fue el mapeo de la retina a la corteza visual (se comportaban como dos grillas, por una parte tenemos a la retina que la podemos pensar como una grilla captando cada celda una porcion de luz, y esta viaja a traves de conexiones sinapticas a otra grilla que es la corteza visual, activando diferentes neuronas ante distintos patrones).

El segundo modelo no trata de explicar detalllles neurobiologicos. Lo que hacemos aqui, es capturar un vector de entrada de cualquier dimension y transformarlo a una matriz de 2 dimensiones. Es decir, es un algoritmo del tipo de vector-coding. Puede ser visto de dos maneras, una de ellas es la idea de auto organizacion, motivada por hechos neurobiologicos, y la otra es tratarlo como un codificador-decodificador cuya motivacion provino del area de teoria de las comunicaciones (Luttrell, 1989, 1991)

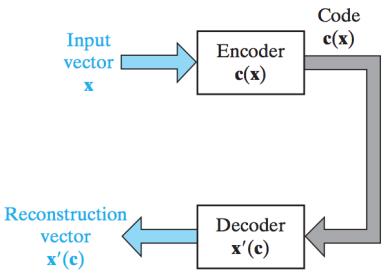


FIGURE 9.5 Encoder–decoder model for describing Property 1 of the SOM model.

2. Desarrollo

2.1. PCA

EL primer modelo utilizado para intentar clasificar los datos será el de Analisis de Componentes Principales. Para ello utilizamremos dos algoritmos basados en aprendisaje Hebbiano y reduciremos las instancias de entrenamiento a 3 dimenciones. Lo que esperamos observar es que aquellas instancias que pertenecen a una misma clase de empresa se encuentran cercas unas de otras, pudiendo observar ïuvesde puntos bien definidas.

2.1.1. Implementación

En particular los algoritmos utilizados serán los de Oja y $Sanger$. Teniendo una complegidad computacional identica y siendo los algoritmos muy similares, lo distintivo entre estos dos metodos es que $Sanger$ ordenará las componentes prinsipales de mayor a menor de acuerdo a sus autovalores mientras que Oja no.

El pseodocodigo utilizado para aprendisaje del algoritmo Oja será:

1: Para toda instancia de entrenamiento, x
2: $y = x.W$
3: $\tilde{x} = y.W^T$
4: $\Delta W = learning\_rate((x - \tilde{x})^T.y$

Algorithm 1: Algoritmo De Oja

Mientras que el de $Sanger$

1: U = Matriz Triangular Superior Con 1s
2: Para toda instancia de entrenamiento, x
3: $y = x.W$
4: $\tilde{x} = W(y^T.U)$
5: $\Delta W = learning\_rate((x^T - \tilde{x}).y$

Algorithm 2: Algoritmo De Sanger

Utilizando el paquete numpy de python es posible traducir este codigo de manera casi exacta y de esa manera aprovechar las optimizaciones matriciales que se realizan sobre los datos.

2.1.2. Experimentación

La metodología de experimentación para este tipo de red será: Primero la entrenaremos con parte del set de datos que nos fue entregado y una vez realizado esto graficaremos los mismo en el espacio marcando con colores cual era la categoría real de cada punto. De esta manera esperamos distinguir nubes de puntos de un mismo color cercanos entre ellos y alejados de aquellos que pertenecen a otras categorías. Como primera instancia quisimos ver que el algormitmo convería realmente a una solución y la manera en que lo realizaba. En el siguiente apartado presentaremos como convergió la solución en cada caso para valores que a priori daban resultados aceptables.

2.1.3. Sanger

Utilizando learning rate igual a 0,1 y 100 epocas, y una matriz inicial de pesos normal con media 0 y varianza $1/\sqrt{(cantidad_neuronas_entrada)}$ entrenamos la red y graficamos los resultados intermedios para $0\,\%, 25\,\%, 50\,\%, 75\,\%$ y $100\,\%$ del entrenamiento. En primer instancia la red comienza completamente desordenada:

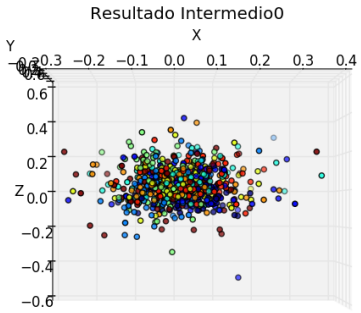


Figura 1: Sin entrenar

Luego de 25 epocas se empieza a observar sierto grado de ordenamiento, por ejemplo, la nube de puntos verdes arriba a la derecha resulta distintiba, lo mismo que una gran agrupación de puntos azules en el medio del grafico.

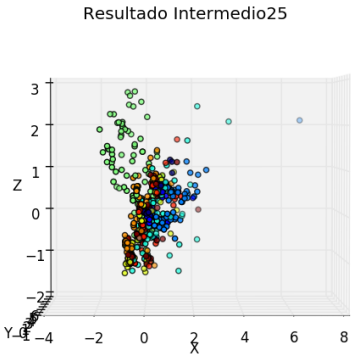


Figura 2: 25 epocas

Para 50 y 75 epocas continua observandose la convergencia del algoritmo, puede observarse como de alguna manera los clusters de puntos rotan en cierta manera quedando distribuidos de manera mas distinitba en el eje x.

(a) 50 epocas (b) 75 epocas

Figura 3: 50 y 75 epocas

Finalmente para 100 epocas puede verse un alto grado de observación de los puntos. De aqui tambien puede observarse que las 3 componentes principales presentan un grado similar de varianza, estando los datos agrupados en $(-2, 2), (-1, 3) y (-2, 2)$ en x, y, z respectivamente. Ademas el punto con $z = -8$ puede obesrvarse un marcado outlier.

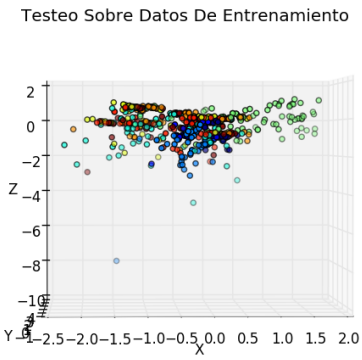


Figura 4: 100 epocas

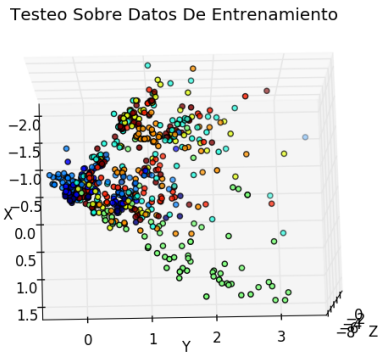


Figura 5: 100 epocas

2.2. Sanger

Realizando un prodecimiento similar para el algormitmo de sanger, podemos observar los siguentes resultados:

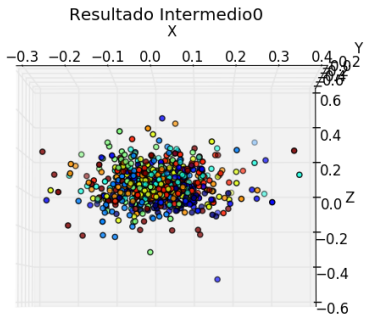


Figura 6: 0 epocas

Comenzando sin ningun tipo de ordenamiento en los datos.

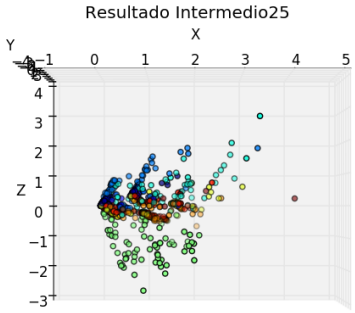
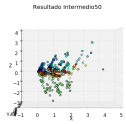
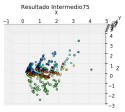


Figura 7: 25 epocas

Aqui ya puede observarse nuevamente sierto ordenamiento de los datos.



(a) 50 epocas



(b) 75 epocas

Figura 8: Resultados Intermedios

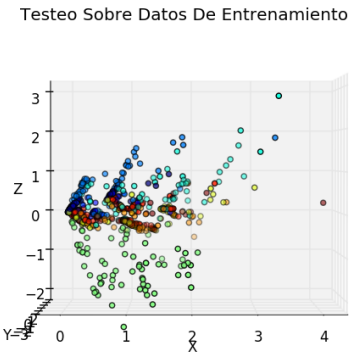


Figura 9: 100 epocas

Nuevamente encontramos aqui nuevamente los datos ordenados

2.3. Overfiting en Sanger y Oja

La proxima experimentación que realizaremos será ver que grado de overfiting realizan estos algoritmos sobre los datos de entrenamiento. Para eso entrenamos con una porción de los datos y utilizaremos el resto para ver graficamente la calidad de los resultados:

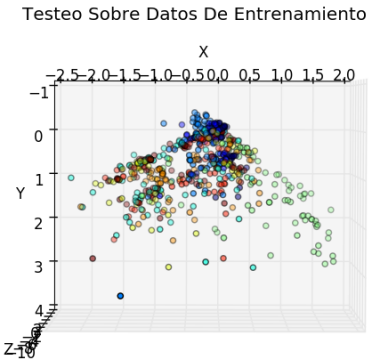


Figura 10: Datos Entrenamiento

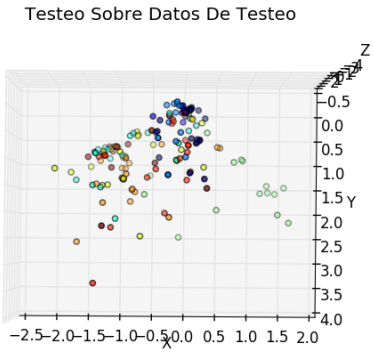


Figura 11: Datos Testeo

Figura 12: Oja

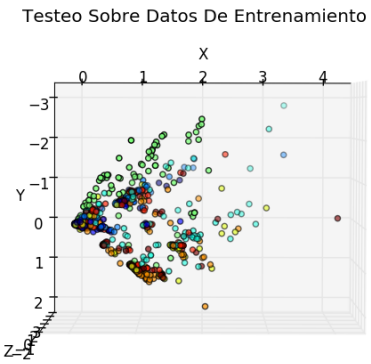


Figura 13: Datos Entrenamiento

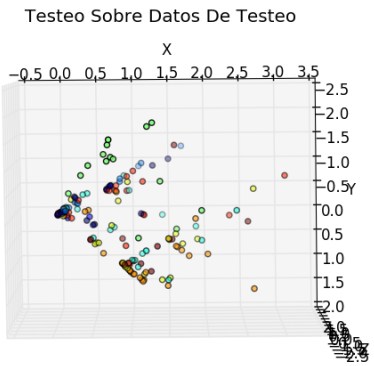


Figura 14: Datos Testeo

Figura 15: sanger

Si bien en algos algoritmos pueden verse puntos que estan fuera del area esperada (por ejemplo en oja, abajo a la izquierda puede verse un punto rojo), concideramos que esto puede deberse al ruido que poseen de manera hinerita los datos. Como una posible solución a los datos, podriían preprocesarse para intentar quitar outliers que empeoren los resultados de los algoritmos haciendolos aprender el ruido de los datos.

2.4. Mapeo de Características

En este apartado construiremos un modelo de mapeo de caracteristicas auto-organizado con la intención de clasificar los documentos en un arreglo de dos dimenciones. Para ello utilizaremos el algoritmo de Kohonen sobre los datos de entrenamiento y una vez que la red haya convergido, intentaremos darle sentido a los resultados.

2.4.1. Implementación

El algoritmo basico utilizado será el visto en clases, que basicamente se divide en:

1: $\tilde{y} = \left\| x^T - MatrizDePesos \right\|$
2: $y = (\tilde{y} == min(\tilde{y})) * 1{,}0$
3: retornar y

Algorithm 3: Activación(x)

1: $j^* = np.nonzero(y)$
2: $D = \Delta(j^*, epoca)$
3: $\Delta pesos = learning_rate(epoca).D(x^T - self.weights)$
4: $MatrizDePesos = MatrizDePesos + \Delta pesos$

Algorithm 4: correccion(x,y)

1: Para cada instancia x de entrenamiento:
2: $y = Activacion(x)$
3: $correccion(x, y)$

Algorithm 5: entrenamiento()

El learning rate comenzará con un valor inicial relativamente alto e irá decreciendo en cada iteración (sin alcanzar al valor cero nunca).

Nuevamente este codigo resulta facilmente adaptable a python.

2.4.2. Convergencia Kohonen

Para esta experimentación utilizamos un learning rate adaptativo igual a $learning_rate_inicial/(1+epocas*learning_rate_proporcional*learning_rate_inicial)$, con $learning_rate_inicial = 0{,}7$ y $learning_rate_proporcional = 0{,}5$

Abajo mostramos la convergencia a priori del algoritmo:

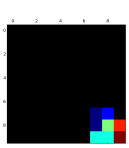


Figura 16: 0 %

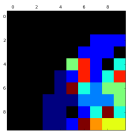


Figura 17: 25 %

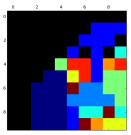


Figura 18: 50 %

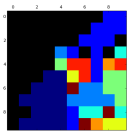


Figura 19: 75 %

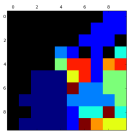


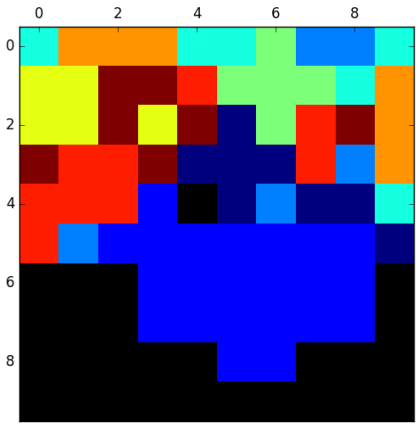
Figura 20: 100 %

Aqui puede verse de manera bastante clara como se van organizando las diferentes categorías.

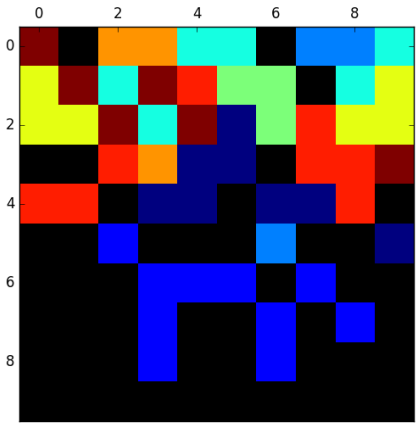
2.4.3. Overfitting en el algoritmo de Kohonen

En este apartado buscamos sacar concluciones sobre posibles overfittings que se puedan realizar sobre los datos a la hora de entrenar un modelo usando kohonen.

Para ello dividimos el training set dado por la catedra en un conjunto uno de entrenamiento y uno de testeo. Luego de entrenar los datos, se obtuvieron estos resultados en ambos sets de datos:



(a) Instancias De Entrenamiento



(b) Instancias De Test

Como puede verse, los datos presentan ciertas dissimilitudes, por ejemplo en el cuadrante $(3, 3)$ en el train set se lo clasifico como que pertenecía a la categoría gris, sin embargo al momento de testeo se clasificó como que pertenecía a la categoría naranja. Esto podría estar causado por dos motivos:

- Los datos presentan ruido, que producen que las clasificaciones no sean perfectas
- O bien, el modelo esta sobreajustando, y memorizando los resultados en vez de generalizar, generando asi respuestas erroneas.

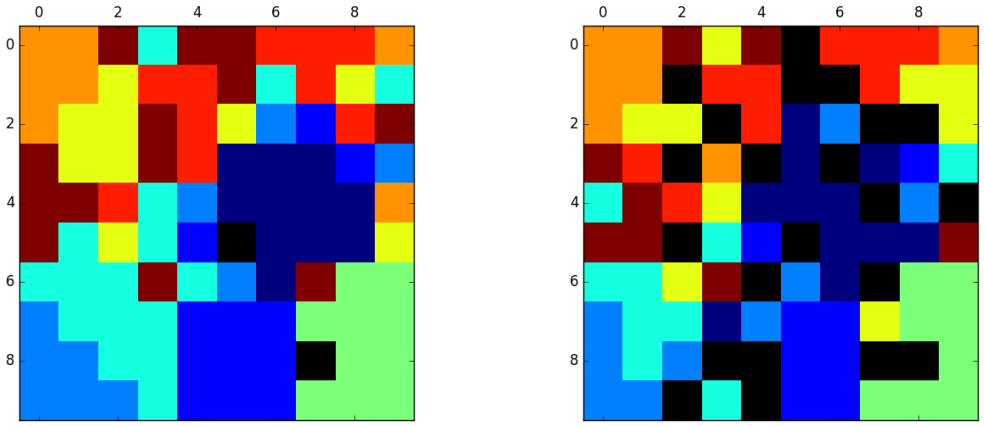
Como prueba adicional a realizar, entrenamos el modelo utilizando otras funciones de learning rate para ver como se comporta en estos casos el overfitting y si es posible mitigarlo de esta manera. Dejando todos los otros parametros constantes, utilizamos:

$$learning\_rate(epocas) = learning\_rate\_inicial * (1 + epocas * learning\_rate\_proporcional) ** - (\alpha)$$

con $learning_rate_inicial = 0{,}7$ y $learning_rate_proporcional = 0{,}5$.

Los resultados fueron los siguientes:

En este caso puede verse un ordenamiento mucho mas fuerte presentando en cada lateral de la figura una categoría diferente. Aun así continua habiendo cuadrantes clasificados de una manera en el set de entrenamiento y de otra en el set de datos. Consideramos sin embargo que estos no son tan grabes como los anteriores y son simplemente casos borde, ninguna instancia parece haber sido clasificada tan lejos de su color en la imagen de entrenamiento.



(a) Instancias De Entrenamiento (b) Instancias De Test

Figura 22: Instancias De Test

3. Detalles De ejecución

3.1. Detalles de ejecución

Para correr el programa simplmente debe ejecutarse:

python main.py numero_ejercicio

Donde *numero_ejercicio* puede ser $1, 2, 3$ siendo oja, sanger y kohonen respectivamente.

Ademas se cuenta con distintos flags opcionales:

- -i Ruta al archivo de entrenamiento
- -o Ruta al archivo donde guardar la red
- -n Ruta al red a utilizar
- -t Ruta al archivo contra el que testear
- -g Graficar Resultados

Por default el programa buscará el archivo de entrenamiento en la carpeta raiz donde se esta ejecutando el programa, en caso de no brindarse un archivo de testing se partirá este mismo en dos partes, se entrenará con una de ellas y se testeará sobre la otra.

En caso de tener habilitado un entorno grafico, el flag $-g$ mostrará en una ventana los resultados de manera visual.

4. Concluciones

PCA: Podemos observar que a medida que aumentamos la cantidad de épocas los datos tienden a esparcirse mas, y esto claramente es debido a que el algoritmo lo que hace es maximizar la varianza. Es decir, en otros términos lo que trata de hacer es lo siguiente: Tenemos 3 ejes en los

cuales podemos representar nuestros datos. Inicialmente todos los puntos estan muy pegados los unos a los otros. Es como si tuviesemos una cámara fotográfica y sacaramos una foto en cierto ángulo viendo todos los puntos unos encima de los otros. A medida que aumentan las iteraciones lo que el algoritmo trata de hacer es cambiar el ángulo de nuestra cámara para poder ver a cada uno de los puntos de manera mas separada posible de sus aledaños para poder en una misma captura, verlos a todos sin que se encimen. Por eso cuando el algoritmo converge, podemos observar a los puntos esparcidos en el espacio, obteniendo la mayor cantidad de información (ya que por ejemplo, si estan todos pegoteados, no tengo forma de separarlos mediante planos por ejemplo para clasificacion, ahora bien si los trato de representar en un sistema de componentes principales quizá estos se separen lo suficiente que me permitan separarlos).

SOM: Podemos ver que claramente en un 25 % de entrenamiento, el algoritmo ya logro su ordenamiento. Lo que resta (es decir las iteraciones restantes) son simplemente para la convergencia del mismo. Esto es fundamentado en el libro de Haykin donde se establece que alrededor de 1000 iteraciones hacen falta para la fase de auto organización de los datos. Luego la fase de convergencia depende fuertemente de la dimensionalidad de nuestro input, pudiendo necesitar una cantidad muy alta de iteraciones. Concuimos a través de la experimentación que el algoritmo pudo hacer una buena clusterización de los datos en distintos grupos bien diferenciados.

Como conclución final podemos apreciar que ambos metodos presentan una alternativa simple y eficaz a la hora de reducir la dimensionalidad de los datos lo que podría resultar muy util al usarlos en conjunto con otras herramientas de aprendisaje automatico, como KNN o PCA. Además el metodo de SOM presenta la posivilidad de .^{en}contrar"posibles clasificaciones subyacentes que no resulten intuitivas a priori sobre los datos.