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
?, ?	?/?	?@?.com

Palabras claves:

TP

Índice

1. Introduccion

En este trabajo, nos disponesmos a utlizar diferentes tecnicas de aprendisaje 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 .ªectivos"(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:

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.ªntre 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.

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.

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)

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 $Sanger$

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.

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

2.1.2. Experimentación

Para la experimentación entrenamos la red con parte del set de datos que nos fue entregado, 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

Con learning rate igual a 0,1 y 100 epocas, y una matriz inicial de pesos normal con media 0 y varianza $1/\sqrt{(cantidad_neuronas_entrada)}$ obtuvimos los siguientes resultados:

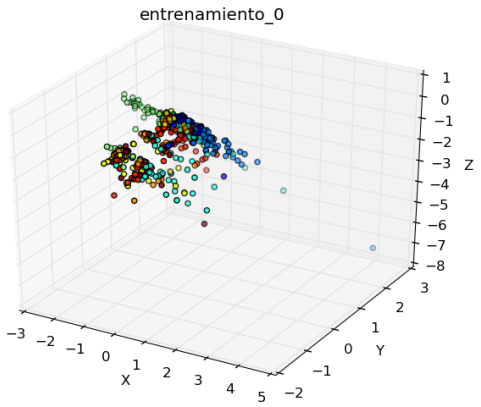


Figura 1: Sin entrenar

En primer instancia, con la matriz sin haber sido entrenada los datos no presentan ningun ordenamiento adecuado.

Luego de 25 epocas se observa:

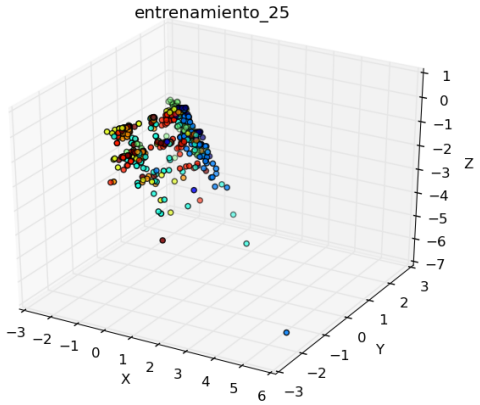
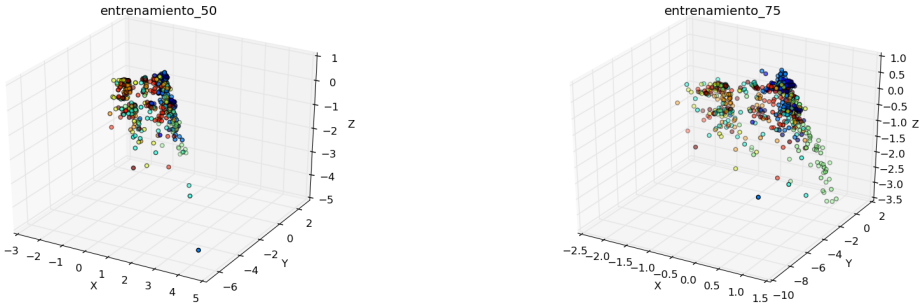


Figura 2: 25 epocas

Ya aquí se presenta cierto ordenamiento...

Para 50 y 75 epocas puede observarse:



(a) 50 epocas (b) 75 epocas

Figura 3: 50 y 75 epocas

Finalmente para 100 epocas puede verse que los datos ya presentan un grado mayor de ordenamiento:

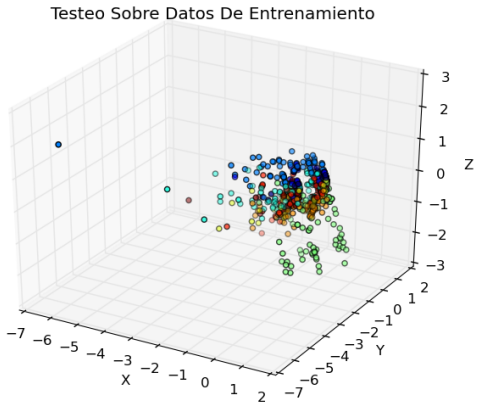


Figura 4: A figure

Ya aquí puede observarse un mayor grado de clusterización. Los puntos verdes y azules estan marcadamente a los costados del graficos.

2.2. Sanger

Realizando algo parecido para el algormitmo de sanger, podemos observar los siguentes resultados:

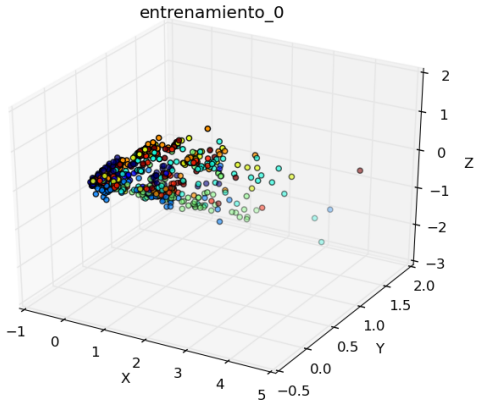


Figura 5: 0 epocas

Ningun ordenaimento.

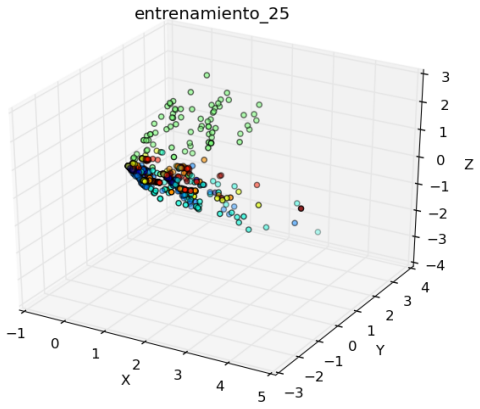


Figura 6: 25 epocas

(a) A subfigure (b) A subfigure

Figura 7: A figure with two subfigures

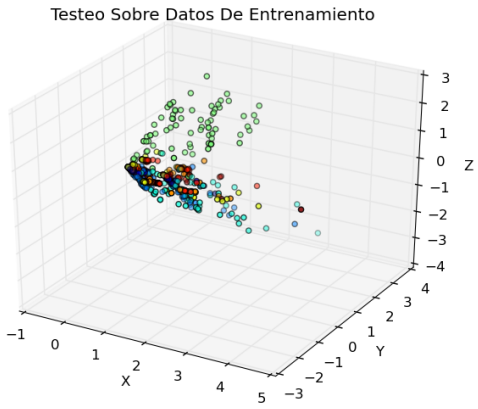


Figura 8: A figure

2.3. 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.

2.3.1. 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)

Convergencia Kohonen

3. Detalles De ejecución

3.1. Detalles de ejecución

Para correr el programa simplmente debe ejecutarse:

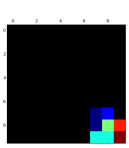


Figura 9: 0 %

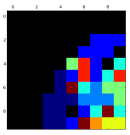


Figura 10: Figura 25 %

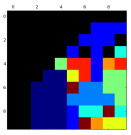


Figura 11: Figura 50 %

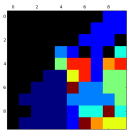


Figura 12: Figura 75 %

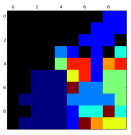


Figura 13: Figura 100 %

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