DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

TP2: Redes Neuronales

Primer cuatrimestre de 2020 Redes Neuronales

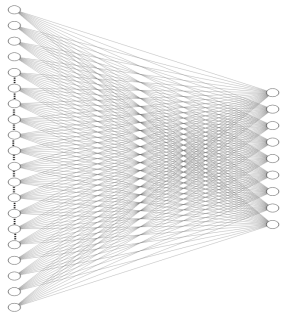
Integrante	LU	Correo electrónico
Goldstein, Brian	027/14	brai.goldstein@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

1. Introducción

En el siguiente informe se expone el modelo de red neuronal utilizado para resolver el problema de reducción de dimensiones planteado en el enunciado en sus 2 variantes, usando las reglas de aprendizaje de Sanger y Oja. También se exponen los métodos explorados para hacer varias el learning rate y lograr así una convergencia en la ortogonalidad total resultante (de la matriz de pesos W) en un tiempo razonable. También se exploraran brevemente las decisiones de implementación tomadas y el preprocesa miento de datos.
Por ultimo se analizaran los resultados obtenidos y se propondrán posibles usos así como limitaciones de esta reducción.
Vale la pena explicitar que todo lo relacionado al como correr el código, esta explicado en detalle en el archivo 'README.md' en el directorio entregado.

2. Modelo Utilizado



El esquema de red neuronal utilizado ya venia implícito en el problema a resolver y en la características de las Redes Hebbianas que se proponía implementar, dado que estas son de una sola copa de profundidad, la capa de input esta determinada por la cantidad de features de nuestro datos y la de output por la cantidad de dimensiones que queremos conservar en la reducción. En nuestro caso esto significo una red de 850 neuronas en la capa de input y 9 neuronas en la de output, sin ninguna capa oculta como se muestra en la figura.

2.1. Reglas de aprendizaje

La red se puede entrenar mediante 2 reglas de aprendizaje diferentes, la regla de Sanger y la Regla de Oja. A continuación se exponen brevemente estas reglas y sus características principales.

Reglas de Sanger:

$$\Delta W = \left(Y.X^T - LT[Y.Y^T].W\right) lr$$

donde lr es el learning rate (con las condiciones de las que se hablaran en la sección siguiente), LT es un operador que pone todos los elementos de la matriz que estén por arriba de la diagonal en 0, convirtiéndola en triangular inferior e Y son las activaciones con el W vigente es decir $Y = WX$.
Reglas de Oja (generalizada):

$$\Delta W = Y.\left(X - Y^T.W\right) lr$$

En donde los parámetros son los mismos que en la regla anterior.

Como puede observarse con un mínimo de álgebra, estas reglas son similares y de hecho la regla de Sanger es una generalización de la de Oja.
Estas reglas pueden reescribirse de muchas formas equivalentes, en este caso fueron expresadas de forma matricial basado en el lo expuesto en paper de *Sanger*, 1989[1] y desde esta fue adaptada la mas simple regla de Oja, con ayuda del material provisto por la materia.
Como esta explicado en el paper de *Sanger*, 1989[1] el algoritmo que aplique la regla de Sanger, provisto que se cumplan las hipótesis: W inicializado de manera aleatoria, $\lim lr = 0$ y $\sum lr = \infty$

garantiza que las filas de la matriz W convergen a las m componentes principales en orden de importancia.
Con la regla de Oja por otra parte, solo se puede afirmar que estos queden de modo tal que describan el subespacio generado por estas m componentes principales, pero no (necesariamente) son las componentes principales excepto en el caso de una única neurona de salida donde el vector que describa el subespacio necesariamente es la componente principal.

2.2. Learning Rate

El algoritmo en ambas variantes (Sanger y Oja) tiene como condición necesaria y suficiente tomar un lr que cumpla:

$$\lim_{t=1}^{\infty} lr(t) = 0 \quad \text{y} \quad \sum_{t=1}^{\infty} lr(t) = \infty \tag{1}$$

Un posible lr que cumple estas condiciones es $lr(t) = \frac{lr_0}{t}$ en donde lr_0 es una constante cualquiera.
Sin embargo en la practica, arrancar con un lr_0 que no sea lo necesariamente pequeño trae problemas de overflow. Por lo experimentado, que tan pequeño debe empezar varia con el tamaño de los datos pero usando todo el dataset provisto, aparecieron errores de overflow con lr del orden de $1e^{-3}$ y mayores. También experimente usando lr fijo.

2.2.1. Learning Rate con memoria

Si bien ambos métodos para elegir el lr funcionaron razonablemente, en muchos casos no se terminaba de lograr la convergencia pues 1 de 2 problemas aparecía:

- El lr funcionaba por unas iteraciones hasta que "quedaba grande", la convergencia total dejaba de mejorar y se quedaba atascada ya que cada paso era demasiado grande (y por tanto el ΔW se pasaba). Esto sucedía principalmente cuando se utilizaba un lr fijo.
- El lr disminuía demasiado rápido lo que hacia que el algoritmo tome pasos muy pequeños cuando todavía podían darse pasos mas grande sin pasarse, lo que hacia que se agote el tiempo sin llegar a una buena solución (incluso dándole tiempos $> 1e^6$). Esto sucedía principalmente cuando se usaba el lr variable.

Para sobrepasar estos problemas implemente una variante de lr variable:
En esta implementación se almacena en una lista los últimos 100 valores de la ortogonalidad obtenida en cada iteración de t anterior.
Luego, en cada iteración se miran los primeros 10 valores de esta lista, es decir los valores mas viejo y se los compara con la ortogonalidad obtenida en la iteración actual.
Si la diferencia entre cada uno de estos valores y la ortogonalidad actual no supera un cierto umbral, es decir si el algoritmo en las ultimas 100 iteraciones prácticamente no mejoro (o mejor pero lo volvió a perder) se determina que se necesita un lr mas chico y se reduce el lr en una orden de magnitud.
Este algoritmo permite mantener un lr grande mientras este este sea efectivo en reducir la ortogonalidad pero en el momento en el que deja de tener efecto se cambia por uno mucho mas chico.
Este esquema de lr es el que mejor funciono y permitió lograr ortogonalidad arbitrariamente pequeña. Fue fijado como condición de parada que esta sea menor a 0,01.

2.3. Criterio de parada

La red dejara de entrenar al llegar a una ortogonalidad menor a 0,01, este numero fue elegido arbitrariamente. Todos los modelos entrenados (aquellos entrenados con toda el dataset y aquellos con particiones) llegan a converger a este valor con el tiempo asignado que por defecto es 20000 iteraciones.

2.4. Preprocesamiento de datos

Para el alimentar el modelo es prerrequisito tener los datos normalizados, es decir centrados en 0 y varianza 1. Para esto se le aplico un StandardScaler a los datos. También se quito la primer columna que guarda la información de las categorías objetivo y se guardo aparte para el posterior análisis, al ser un modelo de aprendizaje no supervisado estas no serán necesarias para el entrenamiento.

3. Resultados

El programa entregado (main.py) luego de entrenar el modelo (de ser necesario) muestra con una animación cada una de las 9 dimensiones, tomadas de a 3 en gráficos donde se codifican las categorías con colores. procedo ahora a hacer un breve análisis de los resultados e incluyo en el informe algunas de las imágenes generadas para dichas animaciones. Estas imágenes (y muchas mas) están en el directorio /plots si se las quiere ver con mas detalle.

3.1. Reducción usando regla de Oja

A continuación imágenes de las dimensiones resultantes utilizando la regla de Oja:

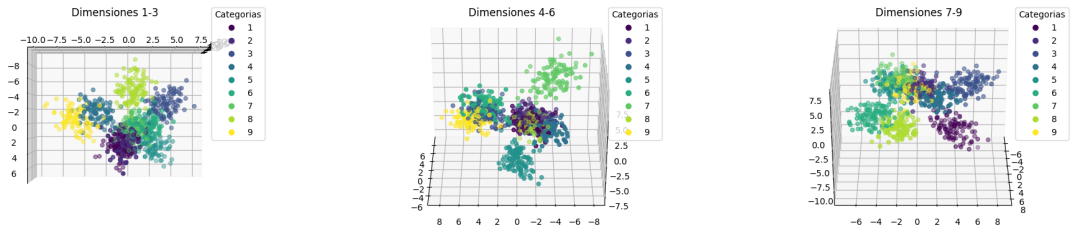


Figura 1: Dimensiones desde los ángulos en los que se visibiliza mejor la separabilidad de las categorías

Como se puede apreciar a simple vista las observaciones parecen acumularse acorde a sus correspondientes categorías. Esto nos da la pauta de que nuestras features están capturando relativamente bien la información que identifica a una categoría.
Se observa que hay ciertas categorías mas fácilmente separables ya que están aisladas, mientras que otras categorías tiene overlapping (en diferentes grados). Se ve que esta separabilidad varia entre las diferentes dimensiones. Todo esto podría potencialmente utilizarse en algún algoritmo de clasificación no supervisada. usando clasificadores que usen diferentes dimensiones para diferentes categorías (o bien y seguramente mejor usarlas todas).
También por ultimo otro posible uso de esta información para interpretar los datos, haciendo

análisis del estilo: si 2 categorías tienen mucho overlaping (acumuladas sobre el mismo espacio), tiene sentido pensar que estan mas relacionadas, mientras que si 2 categorías están muy distantes entre si tiene sentido pensar lo contrario. Aunque hay que tener cuidado con este tipo de análisis pues las distancias entre los diferentes ejes no son necesariamente equivalentes, el caso mas fácil de pensar es si estos ejes fuesen las componentes de pca, la influencia de cada eje seria proporcional al autovalor asociado, en el caso general de Oja seria una combinación de estos.

3.2. Reducción usando regla de Sanger

A continuación imágenes de las dimensiones resultantes utilizando la regla de Sanger:

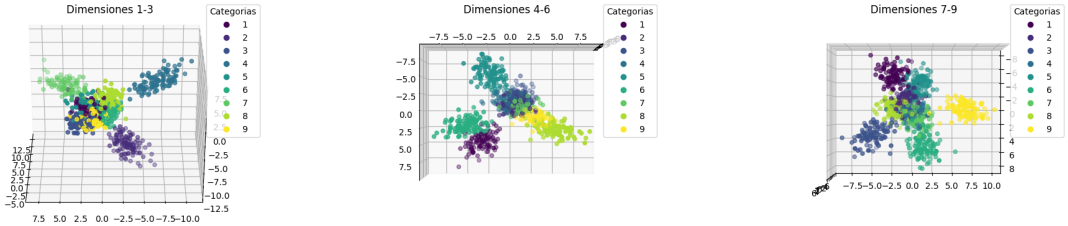


Figura 2: componentes principales desde los ángulos en los que se visibiliza mejor la separabilidad

También como en el caso anterior se acumulan por categorías, aunque en este caso parecería ser un poco mas nítida. Aplican las mismas observaciones en cuanto a la separabilidad de las categorías que con Oja, en este caso se observa aun mas claro que las diferentes dimensiones se pueden aprovechar para separar categorías que en otras dimensiones están aglomeradas, un ejemplo es la categoría amarilla que en las primeras dimensiones esta completamente sumergida en la azul pero en las dimensiones 7-9 esta separada.
También aplican las mismas observaciones respecto a los posibles análisis usando como grado de semejanza entre las categorías a la distancia entre los puntos, en este caso es mas fácil comprender las influencias de cada eje en relación a los demás dado que esta es la descomposición pca y por tanto las distancias pueden traducirse entre ejes multiplicando (o dividiendo) por los autovalores.

3.3. Diferentes instancias

Para evaluar que se obtengan resultados similares con diferentes instancias de entrenamiento escribí un código (split_dataset.py) que particiona el dataset múltiples veces en diferentes tamaños y con una parte de estos datos entrena un modelo y lo guarda como csv en /datasets la otra parte solo la guarda como csv en /datasets para ser usada, ejemplo de estos son el modelo sanger_60_40.sav y los datasets ds_train_60_40.csv y ds_test_60_40.csv, el modelo fue entrenado con el dataset de train y puede probar usarse con el dataset test para ver como funciona con datos que nunca vio. Los resultados obtenidos en términos de gráficos de la reducción son similares mas allá de la menor cantidad de datos.

4. Referencias

[1] Sanger,1989 http://www.cnbc.cmu.edu/~tai/nc19journalclubs/Sanger1989.pdf