**Guía 2 – Anotaciones de ejercicios**

**Redes de base radial y mapas auto-organizativos**

Los ejercicios 1 y 2 son sobre funciones de base radial (en el otro Word de anotaciones) y los ejercicios 3 y 4 sobre mapas auto-organizativos.

**VÍDEO EJERCICIO 3**

<https://drive.google.com/drive/folders/1CK_1RgFgFPJLsWI5UMJGszCgc6Vig9Uy>

**Este es el ejercicio 1 del 2023 – Guía 4**

Comenzamos con un *repaso sobre qué es un SOM*.

Un **SOM** es un mapa auto-organizativo, el cual es una red con un entrenamiento **no supervisado** (no se le indica cuál es la salida correcta que deberían dar los patrones de entrada, sino que van agrupando las categorías de acuerdo a las características o similitudes, no por una salida deseada como los supervisados).

La red recibe patrones de entrada y se auto organiza o acomoda a la estructura de esos patrones con su distribución espacial, eso es lo que llamamos “***mapa topológico***”.

Básicamente cada una de las neuronas va a estar asociada con un vector de pesos, que se puede imaginar como un centroide para esa neurona, y esos pesos tienen la misma dimensión que los patrones de entrada.

*El SOM tiene un entrenamiento* ***no supervisado*** *y* ***competitivo***. Que sea competitivo quiere decir todas las neuronas compiten para ser la que da la salida, es decir, para ser la neurona “ganadora”.

*La neurona ganadora será la que tiene su vector de pesos más cerca del patrón de entradas.*

*¿Cuál es la* ***característica importante*** *que tiene el SOM que permite armar* ***entornos topológicos****?*

Que cada neurona no se mueve sola, sino que la neurona ganadora se va a adaptar para ser más parecida al patrón con el que ganó, pero no se va a mover solo ella, sino también sus vecinas, entonces acá entra el concepto de **vecindad**.

Una vez que detecté la neurona ganadora, además tengo que ver, según el tamaño de vecindad que esté usando, cuales son las neuronas alrededor de la ganadora que también se verán afectadas y deberán actualizar sus pesos. Y entonces *actualizo los pesos de la ganadora y de las vecinas*.

Eso tiene un efecto importante porque *“acopla” la neurona ganadora con sus vecinas y hace que se muevan juntas hacia el patrón con el que se activó la neurona ganadora*, entonces se van todas juntas hacia esa zona, y eso hace *que se copien las relaciones de vecindad en el espacio*, es decir, *dos neuronas vecinas en el arreglo de neuronas que arme, van a tener vectores de pesos que van a representar puntos en el espacio de las entradas que también van a estar cercanos entre sí, porque si esas dos neuronas están cerca en el arreglo de neuronas, se irán moviendo juntas y, por lo tanto, sus vectores de pesos tienen que estar cerca*.

El ***entrenamiento*** es muy sencillo: tomo todos los patrones de entrada, se los muestro al SOM, con cada neurona cálculo la distancia entre el patrón de entrada y el vector de pesos de la neurona, busco cuál es la neurona que tiene la menor distancia y esa es la ganadora. Luego busco las neuronas que estén en la vecindad de la ganadora y actualizo los pesos de la ganadora y de las que estén en su vecindad (con la fórmula de actualización de pesos que dimos en la teoría).

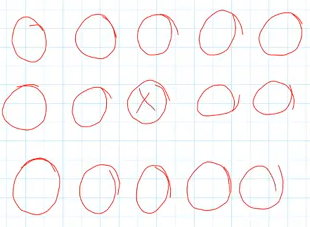
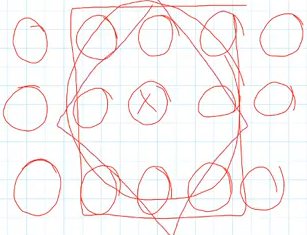
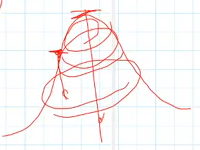
*(Hay dos videos de teoría donde Diego explica el entrenamiento haciendo unos dibujos de todo esto que son buenos para entender cómo explicarlo).*

**Esquema de ejemplo**

Imaginemos que tenemos un **SOM bidimensional**, es decir, que tiene un arreglo de neuronas en dos dimensiones, organizadas como en la imagen de abajo.

Cada neurona tendrá un vector de pesos “w” de la misma dimensión que el vector de entradas, pero recordar que no son pesos que se multiplican a las entradas como en redes anteriores. Todas las neuronas reciben la misma entrada.

Supongamos que para cierto patrón de entrada, la neurona ganadora es la marcada con una “x”, y tengo una vecindad de radio 1. Esa vecindad de radio 1 se podría definir de distintas formas, por ejemplo, en la segunda imagen se marcó tres tipos, una vecindad con el círculo, otra con el cuadrado y otra con el rombo.

Ahí viene la función vecindad, donde dicha función me permite definir un **coeficiente de actualización**, o sea, la fórmula de actualización la multiplico por la función de vecindad, por ejemplo, la función vecindad podría ser una gaussiana (dibujo de arriba a la derecha), entonces la neurona ganadora que estará justo en el medio de esa gaussiana se actualiza con la máxima cantidad que dice la fórmula de actualización (la cima de la gaussiana), pero una vecina que este cerca (punto a la izquierda del centro de la guassiana en el dibujo) se actualiza con un valor más chico, es decir, se actualiza menos que la ganadora. Es un valor que voy a multiplicar por la fórmula de actualización. *Eso me va a garantizar que cuanto más cerca esté una neurona de la ganadora, más se va a actualizar, y que las que están lejos se actualizarán poco.*

Eso teóricamente es lindo, pero *en la práctica es lento*, porque tengo que evaluar la función gaussiana para todas las neuronas del mapa, por ejemplo, si el mapa tienen 10x10 voy a tener que ir actualizando todas las vecinas que correspondan, aunque sea con un valor muy pequeño, y eso hace que la convergencia sea lenta.

Entonces normalmente lo que se hace es usar una *función de tipo “cuadrada”, algo más sencillo, donde hasta cierto radio actualizo todas de la misma manera, y eso simplifica mucho*.

Así que el profe recomendó que usemos la ***función vecindad cuadrada***, que va a ser más rápida, donde usemos una ***vecindad cuadrada*** *y todas neuronas que estén en la vecindad de la ganadora se actualicen igual*, y según el radio de vecindad que puede ir cambiando.

Otra cosa que se suele *simplificar es como calculo la* ***distancia***. Por ejemplo, si en el dibujo de arriba yo quiero medir la distancia entre la neurona que está en (2, 3) es decir, fila 2, columna 3, y la neurona que está en (1, 3), la distancia euclídea a la que están esas dos neuronas seria: d = sqrt( (2-1)^2 + (3-3)^2 ).

Esa fórmula implica restar, elevar al cuadrado y sacar raíces, y eso hay que hacerlo para todas las neuronas, para ver si están dentro del radio, y eso nuevamente es bastante lento, entonces se buscan **simplificaciones**:

-Una simplificación usual es *no usar la raíz*, eso ahorra bastantes cálculos y aplicar o no la raíz no va a cambiar el orden de las distancias, sino que la distancia más corta seguirá siendo la distancia más corta.

-Otra forma de hacerlo es usar *distancias por coordenadas*, por ejemplo, |2-1| y |3-3| directamente, y eso daría lugar a vecindades de tipo rombo como la del dibujo de arriba, que en ese caso esta dibujada con radio 1 pero puede ser más grande. Así que esa también es una simplificación que se usa mucho para acelerar el cálculo.

Entonces, tenemos ***dos tipos de distancias*** implicadas acá:

-Una es la distancia que yo uso para elegir la ganadora, esa se mide entre los vectores de entradas y el vector de pesos, es decir, una distancia a los patrones de entrada.

-Y la otra distancia que tiene que ver con las vecindades, es la distancia entre neuronas, no entre vectores.

Hay que tener cuidado con no confundir eso.

***Repasando*** *lo que dijimos recién*:

-Una simplificación es que usamos *distancia de tipo rombo*, aplicando el valor absoluto de la diferencia en las coordenadas (sin usar la raíz), para hacer la función vecindad más simple y ahorrar cálculos.

-La otra simplificación es que *el peso es el mismo para todas*.

-El *algoritmo* es sencillo, como decíamos, ir recorriendo patrón por patrón y para cada uno ver la ganadora, actualizar la ganadora y su vecindad, e iterar las veces que sean necesarias.

-Lo único complejo es que en la fórmula de actualización se usan dos cosas que cambian a lo largo del tiempo, y *esas* ***dos cosas que cambian a lo largo del tiempo son el tamaño de la vecindad y la tasa de aprendizaje***.

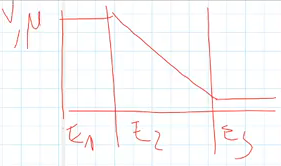
En la teoría vimos que en general se hacen tres etapas de entrenamiento, donde el algoritmo es el mismo, lo único que cambia son los valores que va a tener la vecindad y la tasa de aprendizaje.

En la **primera etapa** se tienen vecindades grandes y velocidad de aprendizaje grande (valores fijos). Esa primera etapa se suele hacer con pocas épocas, solo para que las neuronas se vayan acomodando más o menos hacia la zona del espacio que tienen que ocupar.

Luego, en una **segunda etapa** se va reduciendo en forma lineal el tamaño de la vecindad y de la velocidad de aprendizaje. Es decir, en esta etapa a medida que las neuronas se van acomodando en su zona, las vecindades y la velocidad de aprendizaje se van haciendo más chicas, y así las neuronas se actualizan menos.

Y en una **tercera etapa** o etapa final de ajuste fino, la vecindad solo abarca a la neurona ganadora (no hay vecindad) y la velocidad de aprendizaje es muy chica, haciendo que cada ganadora se ajuste muy bien a su zona (solo se actualiza la ganadora en esta etapa), por eso se le llama etapa de ajusto fino.

Entonces, si hacemos un gráfico y vemos el *tamaño de la vecindad o de la tasa de aprendizaje a lo largo del tiempo*, sería algo como en la imagen de abajo, donde se marcan las tres etapas y vemos que en la primera tenemos valores grandes, en la segunda disminuye de forma lineal y en la tercera se mantiene en un valor pequeño (y la vecindad sería 0).



En el **código**, la forma más fácil de representar eso sería dividiendo el código en **tres for** o while, uno para cada etapa, y en cada etapa usar formas distintas de definir la vecindad y la tasa de aprendizaje.

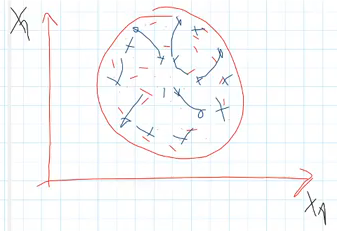
Sino, **otra forma** podría ser hacer esa función que graficamos, que sea para la vecindad y tasa de aprendizaje en función de las épocas, y con **un solo for** iría eligiendo que le toca según las épocas que lleva.

-Todo lo anterior sería con respecto a los aspectos algorítmicos y la implementación, pero ***¿qué es lo que queremos hacer con esta red? ¿Qué deberíamos ir visualizando? ¿Qué debería pasar con las neuronas?***

*Las neuronas deberían copiar la distribución de patrones en el espacio, eso es lo que se llama “****distribución topológica****”.*

Eso quiere decir que, *en zonas del espacio donde haya muchos patrones debería haber más neuronas*, es decir, se van a distribuir las neuronas en las zonas del espacio donde hay patrones, pero además, van a copiar la distribución de esos patrones, donde hay más densidad de patrones debería haber más densidad de neuronas.

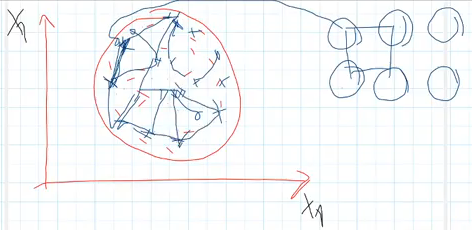
¿Cómo podemos visualizar eso? Lo podemos visualizar si hacemos pruebas en espacios donde podamos graficar, por ejemplo, en R2 o R3. Por ejemplo, en la imagen de abajo graficamos en R2 (con coordenadas x1 y x2), si mis patrones están distribuidos dentro de ese círculo dibujado (los rojos), si inicializo con valores al azar, inicialmente cada neurona tiene un vector del mismo tamaño que los vectores de entradas, entonces los vectores tendrán un w1 y w2. ¿Qué debería ir viendo a lo largo del tiempo? Que esos centroides o medias se irían moviendo hacia ciertos puntos (las rayas azules), que tratan de copiar la distribución de los datos, es decir, las neuronas se irían colocando distribuidas en ese círculo (las “x” azules distribuidas).



Ahora, ¿cómo sabemos que se mantuvieron vecinas? Porque esa es la otra condición que debe pasar, que *neuronas que estaban cerca en el arreglo, que eran vecinas, también estarán en zonas cercanas en el espacio*.

Entonces ¿qué hago? Lo que voy a hacer es un gráfico donde voy a ir uniendo con una recta las neuronas con sus vecinas para comprobarlo.

Entonces lo del “**mapa topológico**” es esto, voy recorriendo las neuronas, grafico su vector de pesos y después uno con rectas las neuronas que son vecinas, y eso me va a hacer una grilla deformable en el gráfico que me muestra las relaciones de vecindad, y vamos a poder verificar si se cumple o no que las vecinas se ubican cercanas en el espacio durante el entrenamiento.



***Volviendo al enunciado del ejercicio*:**

-Cuando nos dice “*que se pueda ver gráficamente el mapa topológico durante todo el proceso*” es eso último que explicamos, la idea es que a medida que van pasando las etapas del entrenamiento, mostremos en esa gráfica donde estaban los patrones de entrenamiento y dónde estaban las neuronas con sus vecinas, y si las unimos con rectas a las vecinas veremos cómo esa grilla se va deformando hasta cubrir todo el espacio.

-Recordar que el enunciado dice usar 1000 épocas como máximo, y las debemos dividir en las tres etapas que explicamos antes.

-Los datos de los dos archivos que vamos a usar dice que fueron generados dentro de las figuras planas de un circulo (circulo.csv) y de una “T” (te.csv), como muestra la imagen del pdf. Eso quiere decir que dentro de la “T” estarían todos los puntitos que representan patrones o datos, y fuera de la “T” no habría datos.

*Deberíamos ir viendo como el mapa topológico a lo largo del tiempo trata de ir adaptándose para tomar la forma de la “T”.*

-Al final del enunciado dice que se pide repetir el entrenamiento con los datos en “T” pero para un ***SOM*** ***unidimensional*** *con la misma cantidad de neuronas*. Supongamos que al comienzo para el resto del ejercicio usamos un SOM de 10x10, ese SOM tiene 100 neuronas, pero ahora el SOM en vez de ser bidimensional podría ser unidimensional, como si fuera una tira de neuronas (una fila de neuronas), entonces si hicimos bien el código, cuando definamos el tamaño de la red si en vez de decirle 10x10 le decimos 1x100 (así tiene la misma cantidad de neuronas, que serían 100 en ese caso) debería hacer una tira de neuronas, y eso debería funcionar.

Lo de 10x10 y 1x100 es un ejemplo, dependen del tamaño que le hayamos definido a la red.

**VÍDEO EJERCICIO 4**

<https://drive.google.com/drive/folders/1CK_1RgFgFPJLsWI5UMJGszCgc6Vig9Uy>

Tenemos una base de datos para clasificación y debemos entrenar el SOM con esos datos. La complejidad del ejercicio es que el SOM es un método no supervisado, nunca usamos la información sobre a qué clase pertenece cada patrón, entonces tenemos que ver cómo podemos usar un SOM como clasificador.

**¿*Cómo podemos usar un SOM como clasificador*?** *(esto se explica bien en el Word de anotaciones de teoría).*

La alternativa más fácil y que vamos a usar es la siguiente:

Imaginemos que tenemos todos los datos y que no usamos la información sobre a qué clase pertenecen, sino que simplemente le doy todos los datos al SOM y él se acomoda según lo que aprende. ¿Qué va a pasar? Si en el espacio hay zonas bien definidas para patrones de distintas clases, lo que debería pasar es que las neuronas que están cerca de esas zonas se activen siempre con patrones de la misma clase, entonces lo que puedo hacer es ***etiquetar las neuronas***, y eso lo hago recorriendo los patrones de entrenamiento y para cada neurona tengo un contador para cada clase, por ejemplo, un vector para cada neurona donde tenga un contador para cada clase, entonces al ir pasando los patrones, cuando gane por ejemplo la neurona 1, me fijo de qué clase era el patrón con el que ganó y aumento en uno el contador correspondiente a esa clase dentro de la neurona 1, y así voy recorriendo todos los patrones, viendo que neurona gana, viendo a qué clase corresponde el patrón y aumento en 1 el contador correspondiente para la neurona ganadora.

De esa manera, cuando termine de pasar todos los patrones, para cada neurona tendré en su vector de contadores cuántas veces salió ganadora para cada clase, entonces decido de qué clase es esa neurona según el mayor valor que tenga en los contadores, por ejemplo, si la neurona se activó 10 veces para la clase 1, 100 veces para la clase 2 y 20 para la clase 3, digo que esa neurona es de clase 2, le asigno a la neurona la etiqueta de “clase 2”.

Entonces después cuando venga un patrón desconocido (datos de prueba), simplemente lo paso, veo cuál es la neurona ganadora y sabiendo eso a dicho patrón le asigno la clase que tiene la neurona ganadora, por ejemplo, si se activa la neurona 1 cuya etiqueta era “clase 2”, ese patrón será de clase 2.

*Y así puedo usar el SOM como clasificador*.

Esa es la forma más sencilla de hacerlo, pero hay *otras formas*, por ejemplo, hacer que cada neurona de como salida la distancia de su centroide al patrón de entrada, y lo que hago es parecido a como hacíamos en una red RBF, de enviar esa salida a una siguiente capa o capa final que se encarga de la clasificación, para que esa capa decida cuál es la clase que corresponde. Pero eso es un poco más complejo, así que usamos la otra forma más sencilla.