

# Práctica de Mapa Auto Organizado

Esther Cuervo Fernández

12 de enero de 2018

## 1. Lectura de fichero

Lo primero que hace el script *smo.m* es leer del fichero de datos normalizados. En *Linux* esta lectura requiere ignorar las líneas vacías del archivo. En *Windows* se ignoran automáticamente. La entrada normalizada tiene valores de 0,1 o 0,9, los cuales transforma en 0 y 1 respectivamente.

Guarda las entradas en una matriz, con una fila por instancia y una columna por componente de la entrada.

## 2. Normalización extendida

A esta matriz se le añade una columna de valores igual a 1, equivalente a considerar una dimensión extra, para evitar el problema de que dos vectores distintos se hagan iguales al normalizar. Tras esto se calcula la norma del vector como:

$$||\bar{x}|| = \sqrt{x_1^2 + x_2^2 + \dots + x_n^2}$$

Siendo  $x_i$  la coordenada  $i$  del vector  $\bar{x} \in \mathbb{R}^n$ . Cada coordenada del vector se divide por esta norma, tal que:

$$\bar{x}^N = \frac{x_i}{||\bar{x}||} \quad \forall i \in [1, n]$$
$$||\bar{x}^N|| = 1$$

Con la matriz de entradas normalizadas, se pasa al entrenamiento del mapa.

## 3. Entrenamiento del mapa

### 3.1. Inicialización de constantes y pesos

En los mapas autoorganizados, el entrenamiento se basa en iteraciones en las que una serie de neuronas ajustan su vector de pesos para acercarse a cada vector de entrada.

Utilizaremos un mapa de  $12 \times 8$  neuronas, y someteremos al mapa a 20 épocas. Se calcula el radio de vecindad inicial necesario para cubrir el máximo de neuronas de la dimensión más pequeña sin que haya overlap, en nuestro caso es  $R = 3$ .

Inicializamos un vector de pesos con valores aleatorios entre  $-5$  y  $5$ , con una fila por cada neurona del mapa, y una columna por cada componente de la entrada original (incluido la añadida para normalización).

Normalizamos los vectores de pesos como hicimos con la entrada.

### 3.2. Entrenamiento

En cada época, recorremos la matriz de entradas, calculando la distancia entre el peso de cada neurona y cada muestra. Para calcular esta distancia utilizamos el *coseno* del ángulo entre ambos vectores.

Este cálculo se basa en el producto escalar de dos vectores. Analíticamente este valor es:

$$\bar{x} \cdot \bar{w} = \sum_{i=0}^n x_i w_i$$

y geométricamente se define como:

$$\bar{x} \cdot \bar{w} = \|\bar{x}\| \|\bar{w}\| \cos(\bar{x}, \bar{w})$$

al normalizar, ambos vectores son unitarios,  $\|\bar{x}\| = \|\bar{w}\| = 1$ , por lo que:

$$\bar{x} \cdot \bar{w} = \sum_{i=0}^n x_i w_i = \cos(\bar{x}, \bar{w})$$

Geométricamente, el *coseno* de un ángulo crece lo más cerca que están los vectores que lo forman, tal que  $\cos(90^\circ) = 0$  y  $\cos(180^\circ) = -1$ . Por tanto el peso más cercano a la entrada será aquel que tenga  $\max(\cos(\bar{x}, \bar{w})) = \max(\sum_{i=0}^n x_i w_i)$ .

En nuestro script los cosenos de una muestra con cada neurona se calculan como la multiplicación de matrices entre el vector  $\bar{x}$  correspondiente a la muestra actual, y la matriz de pesos transpuesto  $\bar{w}^T$ .

La neurona que tenga mayor coseno entre su peso y una entrada será la neurona *ganadora* para esa muestra. Calculamos también  $N(I)$ , que serán las neuronas vecinas a la neurona ganadora dado un radio de vecindad. Nuestro mapa tiene una arquitectura rectangular, por lo que un radio de vecindad  $r$  selecciona un cuadrado de lado  $2r+1$  neuronas alrededor de la neurona ganadora.

Cabe señalar que el mapa es **cíclico**, lo que significa que si el radio de vecindad se sale del mapa, continua aplicándose en el lado contrario. Para implementar esto, se ha utilizado  $fila\_vecina = mod(fila\_calculada, numero\_filas)$ , con la modificación de que si este módulo da 0, la fila es la última, ya que la indexación en *Octave* empieza en 1. Lo mismo se hace con las columnas.

Para  $N(I)$  y la propia neurona ganadora, se modifican los pesos, respecto a la fórmula:

$$\bar{w}(t+1) = \frac{\bar{w}(t) + \alpha(t)\bar{x}}{||\bar{w}(t) + \alpha(t)\bar{x}||}$$

Esta fórmula tiene el efecto de dejar el nuevo peso normalizado.

$\alpha(t)$  es el factor de aprendizaje, que decrece conforme aumentan las iteraciones, respecto a la fórmula:

$$\alpha(t) = \frac{\alpha_0}{1 + t/P}$$

$\alpha_0$  tiene un valor fijado inicialmente, y no cambia durante el programa. En nuestro caso tiene un valor de 20.  $t$  es el número de iteración, que aumenta en uno por cada muestra visitada, y no se resetea a 0 entre épocas.  $P$  es el número de muestras.

Las neuronas fuera de  $N(I)$  no modifican su peso.

Tras procesar todas las muestras, se disminuye el radio de vecindad en 1, y se comienza otra época.

Tras completar todas las épocas, obtenemos los pesos del mapa autoorganizado entrenado.

## 4. Etiquetado por neuronas

Se realiza un etiquetado de las neuronas, con el que obtendremos un mapa con clases asociadas a cada neurona, que nos servirá para la clasificación de nuevas entradas.

Se obtienen las salidas del archivo, y se convierten a un número de 0 a 9, que corresponderá al dígito escrito. Tras esto, se recorren todas las neuronas de la red, y se busca la entrada más próxima, de nuevo utilizando el máximo del *coseno* del peso de la neurona con cada entrada.

Se le da a la neurona la clase que corresponde a la entrada más cercana.

## 5. Verificación

El script abre a continuación el fichero de test, y realiza el mismo procesado que en la entrada y salida del entrenamiento. Tras esto recorreremos las entradas de test, calculando su *coseno* con todas las neuronas, y hallando de nuevo la de peso más cercano. La etiqueta de esta neurona será la clase obtenida para la entrada, que compararemos con la salida deseada que leímos del fichero, añadiendo 1 al número de fallos si no coinciden.

Tras recorrer todas las entradas del fichero de test tendremos la tasa de error para nuestro mapa.

## 6. Utilización de un MLP

Para mejorar la eficacia del mapa autoorganizado, utilizamos un Multi-Layer Perceptron que toma como entrada los cosenos del ángulo entre el peso de cada neurona para cada muestra, utilizando el mapa entrenado para extraer características de las entradas, tras lo cual el perceptrón las clasifica.

### 6.1. Filtrado

Los valores de los cosenos suelen ser muy próximos entre sí, lo que dificulta el reconocimiento del MLP, por lo que se realizará un filtrado para potenciar las diferencias en *smo.m* antes de escribir los ficheros de entrada al MLP.

Se realiza un escalado lineal a los cosenos de cada muestra  $p$  tal que:

$$x_i^p = \frac{x_i^p - \min(x^p)}{\max(x^p) - \min(x^p)}$$

con  $x_i^p$  siendo la componente  $i$  del vector de cosenos  $x$  para la muestra  $p$ .

Tras este escalado se amplifican más aún las diferencias elevando los vectores  $x$  a la 4.

### 6.2. Arquitectura de la red

El MLP usado se trata de una red con una sola capa oculta, que contiene 20 neuronas. La capa de entrada tendrá tantas neuronas como dimensión tiene la entrada, es decir 96 neuronas, y la capa de salida tendrá 10 neuronas, una por cada posible clase.

Por tanto la salida será un vector  $\bar{y}$  con una componente por neurona, y el dígito asignado a la entrada será el índice de la componente más grande del vector.

Se realizarán 2000 épocas, y se utilizará un factor de entrenamiento  $\gamma$  de 0,1 y un factor de inercia  $\alpha$  de 0,1.

### 6.3. Preprocesado

Los datos se introducen al script *mlp.m* a través de dos archivos *.csv* que *smo.m* escribe, uno para el entrenamiento y otro para el test. Estos ficheros contienen una línea por muestra de entrada, y los cosenos del vector peso de cada neurona a esa muestra separados por comas. La última columna contiene la entrada deseada para cada muestra, expresada como un dígito del 0 al 9. Por tanto serán datasets con dimensiones  $270 \times 96$  para el entrenamiento y  $70 \times 96$  para el test.

Transformamos la salida numérica a un vector formado por 0.1s excepto en el índice correspondiente al dígito de la salida, que tendrá un valor de 0.9.

Se realiza una estandarización de los datos de entrenamiento, tal que cada atributo  $a$  de la entrada tiene media  $\mu = 0$  y desviación estándar  $\sigma = 1$ . Esto se realiza mediante la siguiente fórmula:

$$a'_i = \frac{a_i - \bar{a}}{\sigma}$$

Siendo  $a'_i$  el atributo estandarizado en el vector de entrada  $i$ ,  $a_i$  el atributo original en el vector de entrada  $i$ ,  $\bar{a}$  el valor medio de los valores de este atributo para toda la entrada, y  $\sigma$  la desviación estándar.

La manera en la que se estandarizan los datos de test es mediante la misma fórmula, pero se utiliza la media y la desviación estándar del dataset de entrenamiento.

Esta estandarización acelera el aprendizaje del perceptrón. Se han realizado pruebas con este método y con la entrada escalada al rango  $[-1, 1]$  tal como se produce en WEKA, y la estandarización da los mejores resultados.

### 6.4. Entrenamiento

#### 6.4.1. Pesos

Inicializamos dos matrices de pesos,  $ww$  para las conexiones entre la capa de entrada y la capa oculta, y  $ws$  para las conexiones entre la capa oculta y la capa de salida. La conexión de peso  $ww_{ij}$  conecta la neurona de la capa de entrada  $j$  con la neurona de la capa oculta  $i$ . La conexión de peso  $ws_{km}$  conecta la neurona de la capa oculta  $m$  con la neurona de la capa de salida  $k$ . Ambas matrices incluyen una columna extra, que representa la entrada del término bias a la capa oculta/salida respectivamente.

Los pesos se inicializan a valores aleatorios entre  $-5$  y  $5$ .

#### 6.4.2. Fase hacia delante

**Capa oculta** En la fase hacia delante, se introduce una entrada en la red neuronal. Cada neurona de la capa de entrada se encarga de transmitir una componente del vector entrada  $\bar{x}$  a cada neurona de la capa oculta. La salida de cada neurona de la capa oculta se calcula como:

$$u_i^H = \sum_{j=0}^{96} w_{ij}^H x_j + w_{ibias}$$

$$y_i^H = F(u_i^H)$$

Con  $i \in [1, 20]$ . Se ha elegido  $F(x) = \frac{1}{1+e^{-x}}$  como función de activación para la capa oculta.

**Capa salida** Las neuronas de la capa oculta transmiten su salida a las neuronas de la capa de salida. La salida de cada neurona de la capa de salida se calcula como:

$$u_i^S = \sum_{j=0}^{20} w_{ij}^S y_j^H + w_{ibias}$$

$$y_i^S = F(u_i^S)$$

Con  $i \in [1, 10]$ . Se ha elegido  $F(x) = \frac{1}{1+e^{-x}}$  como función de activación para la capa de salida.

#### 6.4.3. Fase hacia atrás

**Capa salida** En la capa de salida, se calcula el error para la muestra, y se retropropaga a través de la red, modificando los pesos. En cada neurona de la capa de salida se calcula el incremento de los pesos de las conexiones a cada neurona de la capa oculta como:

$$\delta_i^S = (d_i - y_i^S) F'(u_i^S)$$

$$\Delta w_{ij}^S = \gamma \delta_i^S y_j^H + \alpha \Delta w_{ij}^S (p - 1)$$

Donde  $d_i$  es la componente  $i$  de la salida deseada,  $y_i^S$  es la salida obtenida de la neurona  $i$  de la capa de salida,  $F'(u_i^S)$  es la derivada de la función usada para calcular  $y_i^S$ ,  $F'(x) = F(x) - (1 - F(x))$ .  $\Delta w_{ij}^S$  es el incremento del peso que conecta la neurona de salida  $i$  con la neurona de la capa oculta  $j$ .  $y_j^H$  es la salida de la neurona  $j$  de la capa oculta.  $\Delta w_{ij}^H (p - 1)$  es el incremento en este mismo peso que se produjo debido a la muestra de entrada anterior. Si esta es

la primera muestra este valor vale 0. Se añade este término para evitar caer en mínimos locales.  $\alpha$  es el factor de inercia y  $\gamma$  el factor de aprendizaje.

**Capa oculta** En cada neurona de la capa oculta se calcula el incremento de los pesos de las conexiones a cada neurona de la capa de entrada como:

$$\delta_i^H = \left( \sum_{k=0}^{20} \delta_k^S w_{ki}^S \right) F'(u_i^H)$$

$$\Delta w_{ij}^H = \gamma \delta_i^H x_j + \alpha \Delta w_{ij}^H (p - 1)$$

Donde  $F'(u_i^H)$  es la derivada de la función sigmoide,  $F'(x) = F(x)(1 - F(x))$ .  $\Delta w_{ij}^H$  es el incremento del peso que conecta la neurona de la capa oculta  $i$  con la neurona de la capa de entrada  $j$ .  $x_j$  es el componente  $j$  de la entrada.

## 6.5. Verificación

Al final de cada época se realiza una pasada por todas las muestras de test, introduciendo cada muestra a la red y calculando la neurona de la capa de salida con mayor valor, que corresponderá al dígito de la clase que la red asigna a esta muestra. Comparamos esta clase con la salida real, y obtenemos el número de fallos, que dividido por el número de muestras del test nos da la tasa de error.

Si este valor está por encima de la tasa de error mínima marcada, se vuelve a entrenar hasta que la tasa disminuya por debajo de este valor, o se alcance el número máximo de épocas.

## 7. Resultados

El etiquetado por neuronas del mapa auto organizado daba resultados bastante buenos, con tasas de acierto alrededor del 92 %. La utilización de un MLP tras el entrenamiento del mapa puede resultar beneficiosa, sobre todo si se utiliza el mapa para reducir un problema con alta dimensionalidad.

En el caso del problema de reconocimiento de dígitos manuscritos, el mapa no reduce la dimensionalidad del problema, si no que realiza una extracción de características. El MLP creado con *Octave* no produce resultados particularmente mejores, aunque la implementación de *WEKA* produce una tasa de error de 0 % con los valores por defecto.