## Parcial 01.

# Implementación del modelo Softmax para el reconocimiento de dígitos escritos a mano.

## Introducción.

En el presente trabajo se construirá un modelo Softmax para el reconocimiento de dígitos escritos a mano, se utilizará las bases de datos *semeion* de imágenes binarias y *mnist* en escala de grises. En *semeion* cada dígito del 0 al 9 representa una clase, cada individuo está representado por 256 pixeles de una imagen binaria $16 \times 16$, donde cero representa un pixel negro y 1 representa un pixel blanco. En *mnist* los datos son ya conocidos.

### Ejercicio.

Lea el archivo *semeion.csv* , separe las primera 256 columnas en una matriz $X$ y la últimas 10 columnas en una matriz $Y$ de 10 columnas. Tome una fila de la matrix binaria $X$ y conviértala en una matriz $16 \times 16$, y dibuje el dígito correspondiente. Repita el proceso eligiendo aleatoriamente 10 filas de $X$ y mostrando el dígito respectivo al que corresponde.

En cada fila de la matriz $Y$ hay un solo uno, la posición que ocupa el 1 en la fila indica el dígito al cual corresponde la imagen.

## Generación de la Red Neuronal para clasificar dígitos.

Cada dígito representa una clase, por lo tanto hay 10 clases. El estímulo que produce cada individuo (fila de $X$) tiene una longitud que depende de la base, en el caso de *semeion*, la longitu es de 256. Vamos a imaginar 10 perceptrones o neuronas (1 por cada clase), cada uno con 257 intérpretes (pesos), para entender la estimulación de cada perceptrón, supongamos que el perceptrón de la clase 4 (correspondiente al dígito 4) tiene los pesos $w_{1,4}$, $w_{2,4}$, $\ldots$,$w_{256,4}$ y $w_{257,4}$, entonces la interacción del iésimo individuo con el perceptrón de la clase 4 es $X_i^* \cdot W_4 = \sum_{j=1}^{257} x_{ij} w_{j4}$, donde $X_i^* = [\ x_{i,1} \quad x_{i,2} \quad \ldots \quad x_{i,256} \quad 1\ ]$la iteracción de cada individuo con cada perceptrón se calcularía a través del producto matricial W*X$^t$, donde cada columna de $X^t$ es un individuo y cada fila de la matriz $W$ son los pesos de un perceptrón.

Tomaremos como función de activación en el $k$-ésimo perceptrón a $f_k(z_k) = \frac{e^{z_k}}{e^{z_0}+e^{z_1}+\ldots+e^{z_9}}$, $k = 0, 1, \ldots, 9$. donde $z_k$ es la interacción producida por un individuo en el $k$-ésimo perceptrón. Al vector columna de probabilidades producido por las activaciones los denominaremos distribución estimada.

### Ejercicio

Programe una función *msoft* de Python, que evalúe el campo vectorial $F(x_1, x_2, \ldots, x_n) = \langle f_1(x_1), \ldots f_n(x_n) \rangle$.

Ahora compararemos el vector de probabilidades $p$ del iésimo individuo con su representación en la iésima fila de $Y$, para ello utilizaremos la función de pérdida $L_i(p) = -\sum_{k=0}^{9} y_{ik} \log p_k$ (entropía cruzada ).

El problema ahora es mínimizar el promedio de las entropias cruzadas producidas por $m$ individuos, es decir la función:

$$E_p = -\frac{1}{m} \sum_{i=1}^{m} L_i(p)$$

## Problemas.

1. Divida la base de datos *semeion* en dos grupos uno de entrenamiento y otro de prueba, tome 1200 elementos de manera aleatoria en el grupo de prueba, imprima los primeros 50 índices obtenidos de este grupo.

2. Programe una función *mgrad* para calcular el gradiente de $Ep$ con respecto de los pesos. Recuerde aplicar la regla de la cadena.

3. Programe un función *redn* en Python que use el método del gradiente descendiente, tomando subconjuntos (bunch o batch) de tamaño 50 y 50 épocas por defecto, para encontrar el conjunto de pesos que minimizan $E_p$. Realice experimentos variando el número de épocas hasta conseguir porcentajes de imágenes bien clasificadas mayores o iguales al 90% en cada clase sobre el conjunto de entrenamiento. La función debe devolver los pesos y el vector de pérdida total en cada iteración.

4. Utilice los pesos obtenidos en el numeral anterior para predecir la clase correspondiente sobre el conjunto de datos de prueba, reporte el porcentaje de bien clasificados por el modelo en cada clase, también reporte los porcentajes de mal clasificados en cada clase (validación cruzada).

5. Haga un gráfico de los valores de pérdida en cada iteración.

6. Programe un función *redna* en Python que use el método del gradiente descendiente, tomando subconjuntos aleatorios (bunch o batch) de tamaño 50 en cada iteración, para encontrar el conjunto de pesos que minimizan $E_p$. Realice experimentos variando el número máximo de iteraciones hasta conseguir porcentaje de imágenes bien clasificadas mayores o iguales al 90% en cada clase sobre el conjunto de entrenamiento. La función debe devolver los pesos , el vector de pérdida total en cada iteración y graficar la función de perdida de manera dinámica (hacer el gráfico en cada iteración).

7. Utilice los pesos obtenidos en el numeral anterior para predecir la clase correspondiente sobre el conjunto de datos de prueba, reporte el porcentaje de bien clasificados por el modelo en cada clase, también reporte los porcentajes de mal clasificados en cada clase (validación cruzada).

8. Repita los numerales desde el dos hasta el siete para los datos mnist de entrenamiento y de prueba dados. Haga iteraciones con subconjuntos de tamaño 500 y 50 épocas, en el caso aleatorio tome conjuntos de tamaño 500 también.

9. Comente todos sus resultados.