Apuntes Semana 7, Jueves 21 de marzo de 2024

Alejandra González Fernández
Escuela de Ingeniería en Computación
Tecnológico de Costa Rica
Cartago, Costa Rica
21/04/2024

I. Clasificación de MNIST

El conjunto de datos MNIST es un conjunto compuesto por 10 clases, que van del número 0 al 9, con dígitos escritos a mano.

Características:

- Originalmente, las imágenes tienen una resolución de 128x128 píxeles.
- Se transforman a 28x28 píxeles, lo que resulta en 784 características.
- Contiene 60,000 muestras.
- 1 channel

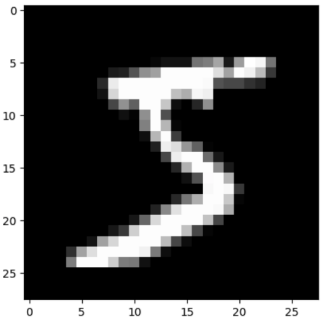

Fig. 1: MNIST Ejemplo
Source: [1]

II. Regresión logística (Clasificación binaria)

La regresión logística nos proporciona una herramienta para determinar si un evento ocurre o no, en este caso, si una imagen corresponde al número 5 o no. Sin embargo, la regresión logística tiene limitaciones cuando se trata de abordar problemas de múltiples clases, ya que es más adecuada para problemas binarios. Para resolver problemas multiclase, es necesario buscar alternativas más adecuadas.

A. ¿Cómo podemos pasar esta imagen como entrada a una regresión logística?

Para pasar una imagen como entrada a un modelo de regresión logística, podemos utilizar un enfoque conocido como *flattening* de la imagen. Esto implica convertir la matriz de píxeles de la imagen en un único vector plano. Por ejemplo, si la imagen es de tamaño 28x28 píxeles, tendríamos originalmente una matriz de 28 filas y 28 columnas. Al aplanar la imagen, convertimos esta matriz en un vector unidimensional de longitud 784 (28x28). Finalmente, este arreglo lineal de 784 características, donde cada una representa el valor de cada píxel de la imagen (en un rango de 0 a 255), se convierte en la entrada de la regresión logística.

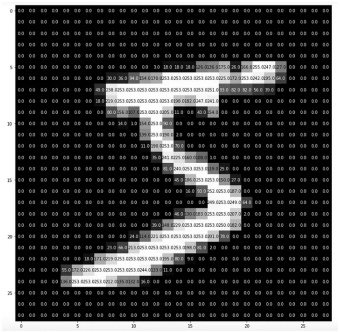

Fig. 2: MNIST Ejemplo Píxeles
Source: [2]

B. ¿De qué tamaño debería ser el input layer?

El tamaño del input layer en este caso debe ser igual al número de características en el arreglo lineal de entrada, que es 784. Cada una de estas características representa un píxel en la imagen. Por lo tanto, para la regresión logística que estamos aplicando, cada píxel se mapea a una entrada en el input layer.

En la regresión lineal que precede a la regresión logística, estos píxeles se utilizan como las variables independientes x para predecir la variable dependiente y. Cada uno de los píxeles tiene su propio peso w y se suma a un término de sesgo b .

$$f_{w,b}(x) = wx + b \tag{1}$$

Luego, esta salida de la regresión lineal se pasa a través de la función de activación logística para determinar la probabilidad de que la imagen pertenezca a la clase 0 o a la clase 1.

III. Regresión logística multinomial

Para la regresión de múltiples clases, consideramos un ejemplo donde utilizamos 10 regresiones logísticas, cada una encargada de determinar si la imagen corresponde a un número del 0 al 9. Con esto, podemos construir un *one-hot* vector que

representará las clases y que será utilizado como las etiquetas y.

Cada clase tiene una codificación en bits que corresponden a las salidas de las regresiones lineales o a las capas de salida.

TABLE I: *One-hot* vector

class	***one-hot codification***
0	1000000000
1	0100000000
2	0010000000
3	0001000000
4	0000100000
5	0000010000
6	0000001000
7	0000000100
8	0000000010
9	0000000001

Características:

- Se envían todos los píxeles de la imagen a cada una de las regresiones logísticas.
- Cada regresión logística se especializa en clasificar un número específico.
- El resultado general es un *one-hot* vector que indica la clase a la que la imagen más probablemente pertenece, basada en las salidas de las regresiones logísticas.
- Este vector tiene un 1 en la posición correspondiente a la clase predicha y 0 en las demás posiciones.

IV. Conversión de vectores a matriz

Para la conversión de vectores a matriz, en lugar de calcular cada regresión lineal de forma vectorial, utilizaremos matrices para realizar una sola operación en lugar de N operaciones individuales, donde N es el tamaño de la capa siguiente. Esto se basa en conceptos de álgebra lineal.

TABLE II: Ejemplo Tabla Matriz

$w_{0,0}$	$w_{0,1}$	$w_{0,2}$	$\cdots$	$w_{0,n-1}$	$w_{0,n}$
$w_{1,0}$	$w_{1,1}$	$w_{1,2}$	$\cdots$	$w_{1,n-1}$	$w_{1,n}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
$w_{j,0}$	$w_{j,1}$	$w_{j,2}$	$\cdots$	$w_{j,n-1}$	$w_{j,n}$

Donde:

- j = el tamaño de la siguiente capa
- n = el tamaño de los features.

En el ejemplo específico del dataset de MNIST, en lugar de tener solo vectores separados, tendremos una sola matriz donde:

- La dimensión de las filas es el tamaño de la siguiente capa (en este caso, 10 para las 10 clases).
- La dimensión de las columnas son los features (en la primera capa, 784).

Entonces, en nuestro caso, el tamaño de la matriz debería ser 10x784.

TABLE III: Ejemplo Tabla Matriz MNIST

$w_{0,0}$	$w_{0,1}$	$w_{0,2}$	$\cdots$	$w_{0,783}$	$w_{0,784}$
$w_{1,0}$	$w_{1,1}$	$w_{1,2}$	$\cdots$	$w_{1,783}$	$w_{1,784}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
$w_{10,0}$	$w_{10,1}$	$w_{10,2}$	$\cdots$	$w_{10,783}$	$w_{10,784}$

V. ¿Qué sucede con el bias?

Antes, para una única regresión logística, se tenía un solo bias que era un escalar. Sin embargo, ahora, al tener múltiples regresiones logísticas, ya no se tiene ese concepto de un único escalar para el bias. Cada regresión logística tiene su propio bias, y estos pueden representarse con un vector.

A. Ejemplo 2 operaciones

1. Dados los vectores y los escalares:

$$\begin{aligned}\mathbf{x} &= [3,4,5,6]\\ \mathbf{w}_1 &= [3,2,4,5]\\ b_1 &= 2\\ \mathbf{w}_1 \cdot \mathbf{x} + b_1 &= [3,4,5,6]\cdot[3,2,4,5] + 2 = 67 + 2 = 69\end{aligned}$$

$$\mathbf{sigmoid}(\mathbf{w}_1 \cdot \mathbf{x} + b_1)$$

$$\begin{aligned}\mathbf{x} &= [3,4,5,6]\\ \mathbf{w}_2 &= [4,3,2,1]\\ b_2 &= 3\\ \mathbf{w}_2 \cdot \mathbf{x} + b_2 &= [3,4,5,6]\cdot[4,3,2,1] + 3 = 40 + 3 = 43\end{aligned}$$

$$\mathbf{sigmoid}(\mathbf{w}_2 \cdot \mathbf{x} + b_2)$$

B. Mismo ejemplo pero con matrices

$$\begin{aligned}\mathbf{x} &= [3,4,5,6]\\ \mathbf{W} &= \begin{bmatrix}3 & 2 & 4 & 5\\ 4 & 3 & 2 & 1\end{bmatrix}_{\text{(regression, feature)}}\\ \mathbf{b} &= [2,3]\end{aligned}$$

La operación se calcula como:

$$\mathbf{x}\mathbf{W}^\top + \mathbf{b}$$

$$\text{sigmoid}(\mathbf{x}\mathbf{W}^\top + \mathbf{b})$$

Si se puede hacer eso con las matrices de peso y con los sesgos se puede hacer lo mismo con los x, que en vez de estar multiplicando un x a la vez se podría tener un batch y procesar varios samples a la vez donde cada uno de esos samples va a tener 784 features.

$$WX + b \tag{2}$$

Se podría tener una expresión como $WX + B$ donde X y W son matrices. X representa una matriz de inputs, W una matriz de pesos sin bias y b el vector que contiene un bias por cada regresión.

VI. Clasificación binaria

- Capa de entrada: Es la primera capa, la capa de features.
- Capa intermedia: Esta capa es donde se lleva a cabo el procesamiento principal de la red neuronal. Puede haber una o varias capas intermedias, dependiendo de la complejidad del problema y de la arquitectura de la red neuronal.
- Capa de salida: No depende de la entrada del modelo; su entrada son las salidas de cada una de las regresiones de la capa anterior.

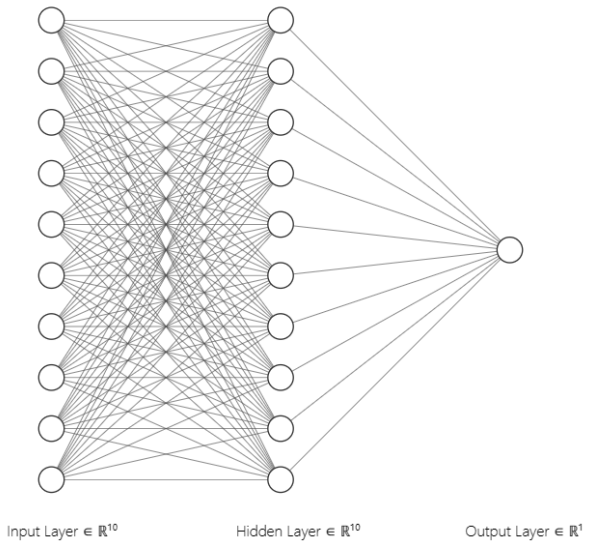

Fig. 3: Red Neuronal Ejemplo

Lo que llegaría a la última capa no representa los píxeles de la imagen, sino las salidas de las regresiones lineales de la capa intermedia. Esto nos proporciona la capacidad de tener funciones no lineales entrelazadas, cada una con las anteriores, lo que permite la no linealidad. Esto nos permite resolver problemas más complejos de los que se pueden resolver linealmente.

Otras características:

- Las capas intermedias y el número de neuronas por capa intermedia son decisiones del desarrollador (hiperparámetro).
- Las capas deben ser diferenciables para poder ser optimizadas.

VII. Antecedentes

Las regresiones logísticas fueron propuestas por Frank Rosenblatt.

Frank Rosenblatt fue un destacado psicólogo estadounidense en el campo de la inteligencia artificial. También lideró los primeros trabajos sobre perceptrones, que culminaron en el desarrollo y construcción del hardware del Perceptrón Mark I en 1960. Este fue fundamentalmente el primer ordenador que podría aprender nuevas habilidades mediante ensayo y error, utilizando un tipo de red neuronal que simulaba el proceso de pensamiento humano [3].

Las expectativas eran muy altas. Sin embargo, cuando el Perceptrón finalmente se presentó en julio de 1958, solo logró completar una tarea: reconocer de forma independiente, después de cincuenta intentos, qué tarjeta perforada estaba marcada a la izquierda y cuál a la derecha. De hecho, hoy en día se considera que el auge y la caída del Perceptrón fueron la causa del primer ”invierno de la IA”, una fase durante la cual el entusiasmo por este campo se enfría y la investigación prácticamente se detiene [4].

A. *Invierno de la IA*

El ”Invierno de la IA” fue un período en el que las investigaciones en inteligencia artificial experimentaron un declive significativo. Minsky y otros investigadores señalaron varios problemas durante este tiempo, incluyendo:

- Los requisitos computacionales eran muy elevados para la época.
- No se podía resolver un problema básico como el operador lógico XOR con las técnicas disponibles en ese momento.
- Como resultado de estos desafíos y limitaciones, las investigaciones en inteligencia artificial se detuvieron o disminuyeron considerablemente durante este período.

B. *Problema con el XOR*

- No puede ser modelado por una regresión logística o un perceptrón debido a su naturaleza no lineal.
- En los casos del OR y el AND, es posible realizar una separación lineal entre las clases, pero en el caso del XOR, no había forma de separarlo linealmente, lo que lo hacía especialmente desafiante para los enfoques lineales como las regresiones logísticas y los perceptrones.

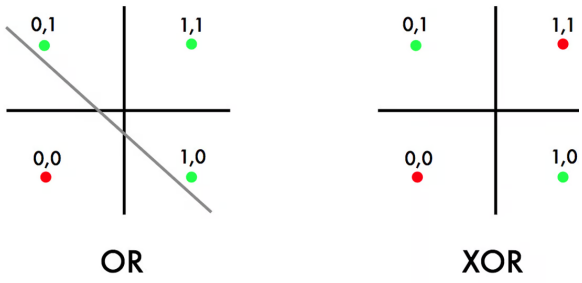

Fig. 4: OR and XOR
Source: [5]

Sin embargo, este problema fue resuelto por las redes neuronales, específicamente porque tienen la capacidad de modelar problemas no lineales.

VIII. Función de activación

- En la regresión logística, la función de activación se llamaba función no lineal y comúnmente era la función sigmoidal.
- La función de activación determina si una neurona se activa o no, dependiendo de la señal recibida.
- Esta función puede permitir el paso de información, transformarla o bloquearla, dependiendo de su diseño.
- Además de la función sigmoidal, existen otras funciones de activación que se comportan de manera no lineal. Cada una tiene sus propias propiedades y aplicaciones en diferentes contextos de redes neuronales.

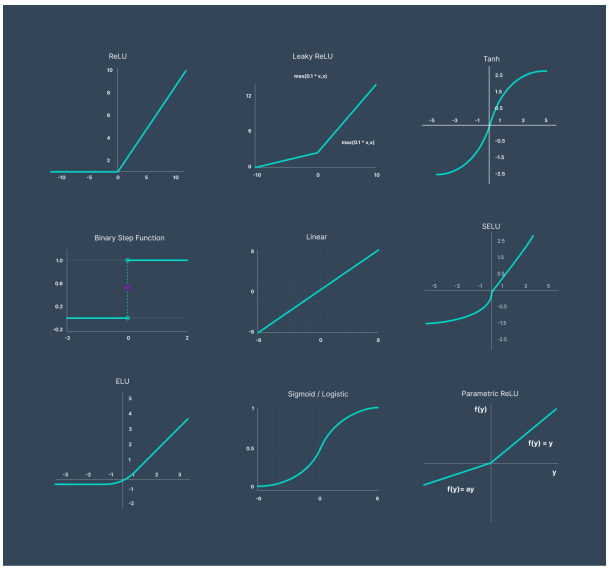

Fig. 5: Funciones de activación
Source: [6]

Algunas características son:

- ReLU (Rectified Linear Unit): Si el valor de entrada es menor que 0, la función se comporta como 0; de lo contrario, elige el máximo entre 0 y x.
- Leaky ReLU: Similar a ReLU, pero cuando los valores de entrada son menores que 0, en lugar de ser 0, son multiplicados por un pequeño valor (como 0.1) para permitir un ligero aumento en la pendiente.
- Tangente Hiperbólica: Esta función está acotada entre -1 y 1 y es similar a la función sigmoidal pero con un rango mayor.

En las capas intermedias de la red neuronal, se prefiere comúnmente ReLU debido a su capacidad para optimizar mejor la red. Esto se debe a consideraciones como las derivadas y el problema del desvanecimiento del gradiente. El algoritmo de gradiente descendente se utiliza en el entrenamiento de redes neuronales, donde se realizan iteraciones sobre los parámetros del modelo de manera proporcional al valor negativo del gradiente en el punto actual. La propagación hacia atrás se emplea para calcular el gradiente y corregir los parámetros del modelo [7].

La selección de la función de activación adecuada es crucial, especialmente para la capa final de la red, donde su elección puede influir significativamente en el rendimiento del modelo.

IX. Perceptrón Multicapa MLP

La capa de entrada tiene un vector de 784 elementos. Lo que se va a hacer es calcular el sigmoid de x con la capa de pesos W y los vectores B. Al final, se obtiene un vector con 10 elementos, que representan las salidas de cada una de las neuronas de $h(0)$ respectivamente.

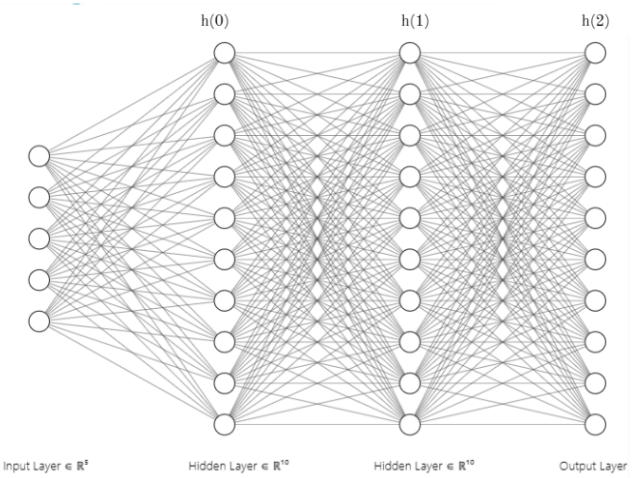

Fig. 6: Red neuronal

MLP:

$$h(0) = \text{sigmoid}(XW^0 + b^0)$$

Ahora, $h(0)$ es la entrada de la siguiente capa. Se utilizan los cálculos hechos en $h(0)$, se multiplican por la siguiente matriz de pesos W (de la siguiente capa) y se les suma los respectivos vectores B.

$$h(0) = \text{sigmoid}(XW^0 + b^0)$$
$$h(1) = \text{sigmoid}(h^{(0)}W^1 + b^1)$$

Y así sucesivamente hasta llegar a $h(n)$, que representa la capa final. Cada capa se calcula utilizando los resultados de la capa anterior, multiplicando por la matriz de pesos correspondiente W y sumando los vectores B respectivos.

$$h(0) = \text{sigmoid}(XW^0 + b^0)$$
$$h(1) = \text{sigmoid}(h(0)W^1 + b^1)$$
$$h(2) = \text{sigmoid}(h(1)W^2 + b^2)$$
$$h(n) = \text{sigmoid}(h(n-1)W^n + b^n)$$

A. Salida independiente

En este punto, en nuestra capa de salida, tenemos probabilidades independientes. Por lo tanto, no hay una distribución de probabilidad. Es necesario que de alguna manera las salidas se comporten como una distribución, como una distribución de probabilidad de observar cada uno de los diez elementos.

TABLE IV: Output layer de la última capa

Output layer
0.05
0.05
0.06
0.04
0.1
0.5
0.05
0.05
0.05
0.05

Esto se logra transformando las probabilidades en un *one-hot* vector.

***one-hot* vector**
0
0
0
0
0
1
0
0
0
0

B. Capa de salida

$$h(n) = g(h(n-1)W^n + b^n)$$

- Donde $g(x)$ no necesariamente es la función sigmoide.
- Función no lineal.

X. Función de costo

- Debemos optimizar los parámetros W y b de cada una de las neuronas.
- ¿Cómo hacemos?

$$\frac{\partial L}{\partial w_{ij}}$$

donde $j =$ capa y $i =$ neurona.

A. ¿Cuántas capas?

Varía según el problema y la complejidad, pero se suelen usar múltiples capas para aprender representaciones complejas.

B. ¿Cuántas neuronas por capa?

Depende de la dimensionalidad de los datos y la complejidad del problema; se experimenta para encontrar la mejor configuración.

C. ¿Por qué una jerarquía?

Permite que el modelo aprenda desde lo general a lo específico, capturando patrones complejos y permitiendo un ajuste fino para tareas específicas.

XI. Maldición de la dimensionalidad

La maldición de la dimensionalidad es un fenómeno en el que, al aumentar el número de dimensiones en un espacio de datos, la densidad de los datos disminuye drásticamente. Esto puede dificultar la búsqueda de puntos óptimos, como encontrar una bolita roja en un edificio de dimensiones cada vez mayores.

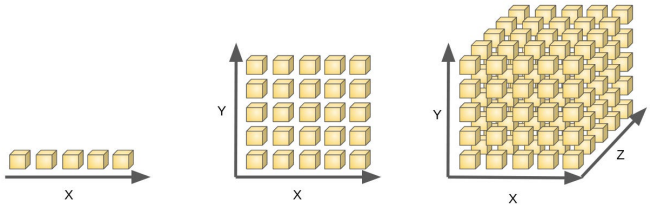

Fig. 7: Gráfico de diferentes dimensiones
Source: [7]

- A mayor cantidad de dimensiones, aumenta la complejidad
- Aumenta la computabilidad.
- Difícil encontrar patrones.
- Reducción de dimensionalidad (PCA)

Las redes neuronales son una solución que aborda este problema.

A. Mapas de activación

Los mapas de activación proporcionan un ejemplo de cómo se ve la salida de una capa de neuronas, especialmente después de reducir la dimensionalidad. En el caso de las capas de convolución, la información tiende a pasar de características generales a específicas a medida que avanzamos en las capas.

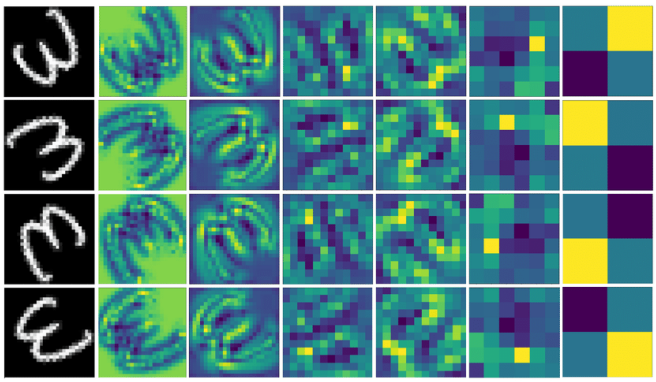

Fig. 8: Mapa de activación MNIST
Source: [8]

XII. Bibliografía

[1] Brownlee, J. (2021) How to develop a CNN for mnist handwritten digit classification Available at: https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/ (Accessed: 28 March 2024).

[2] Mendels, G. (2023) Real-time numbers recognition (MNIST), Comet. Available at: https://www.comet.com/site/blog/real-time-numbers-recognition-mnist-on-an-iphone-with-coreml-from-a-to-z/ (Accessed: 28 March 2024).

[3] Frank Rosenblatt (2023) Wikipedia. Available at: https://es.wikipedia.org/wiki/Frank_Rosenblatt (Accessed: 28 March 2024).

[4] Signorelli, A.D. (2023) Conoce a Frank Rosenblatt, Creador de la Primera Inteligencia artificial, Wired. Available at: https://es.wired.com/articulos/conoce-a-frank-rosenblatt-creador-de-la-primera-inteligencia-artificial (Accessed: 28 March 2024).

[5] Ahire, J.B. (2020) Demystifying the XOR problem, DEV Community. Available at: https://dev.to/jbahire/demystifying-the-xor-problem-1blk (Accessed: 28 March 2024).

[6]Baheti, P. (2021) Activation functions in neural networks V7. Available at: https://www.v7labs.com/blog/neural-networksactivationfunctions (Accessed: 28 March 2024).

[7] Fuentes, J. (2020) The independent sentinel 13, 13 - by Javier Fuentes. Available at: https://theindependentsentinel.substack.com/p/the-independent-sentinel-13 (Accessed: 28 March 2024).

[8] Implicit Equivariance in Convolutional Networks - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Extended-results-for-the-MNIST-feature-map-representations-shown-in-the-main-paper-We_fig1_356631956 [accessed 28 Mar, 2024]