

# Apuntes sobre IA

Sergio de Mingo

2 de febrero de 2026

## Índice

1. Reconocimiento de patrones	1
-------------------------------	---

## 1. Reconocimiento de patrones

Hasta ahora, tu red solo ha tenido que distinguir entre 4 combinaciones. Pero, ¿qué pasa cuando la entrada no son dos bits (0 o 1), sino una imagen de 28x28 píxeles de un número de 0 a 9 escrito a mano? El dataset MNIST es el “Hola Mundo” del Deep Learning. Son imágenes de dígitos del 0 al 9. MNIST Es una extensa colección de base de datos que se utiliza ampliamente para el entrenamiento de diversos sistemas de procesamiento de imágenes. La base de datos MNIST consta de 60.000 imágenes de entrenamiento y 10.000 imágenes de prueba. Ahora nuestra entrada  $X$  ya no es una matriz de  $4 \times 2$ . Ahora cada imagen se aplana en un vector de 784 números (28 píxeles x 28 píxeles). Si procesamos un batch de 64 imágenes,  $X$  será  $[64 \times 784]$ . La salida  $Y$  ya no es un solo valor. Ahora necesitamos saber la probabilidad de que sea un 0, un 1, un 2... hasta el 9. Por tanto, la salida tiene 10 neuronas.

En el XOR usamos la Sigmoid, pero para redes más profundas existe un problema importante con esta función: El **gradiente desvaneciente** o *Vanishing Gradient*. Esto se produce al derivar la sigmoid. En estos casos, el valor máximo es 0.25. Si multiplicas muchos números decimales tan pequeños en la regla de la cadena, el error se vuelve tan pequeño que las primeras capas nunca aprenden porque a ellas el delta del error siempre llega como 0. La solución a este problema será usar la función ReLU o *RectifiedLinearUnit*. Su fórmula no puede ser más simple y se muestra a continuación. Si es positivo, deja pasar el valor del error tal cual. Esto mantiene los gradientes «vivos» y hace que el entrenamiento sea muchísimo más rápido.

$$f(x) = \begin{cases} 0 & \text{si } x \leq 0 \\ x & \text{si } x > 0 \end{cases}$$

Otro problema ahora es que en la salida necesitamos varias neuronas, en concreto 10. Así cada una nos devolverá una probabilidad de que el patrón de la entrada es el número asociado a ellas. En el XOR, la salida era un valor entre 0 y 1. En clasificación de números, queremos que la red nos diga: «Estoy un 80 % seguro de que es un ‘5’, un 15 % de que es un ‘3’ y un 5 % de que es un ‘6’». Para esto usamos la función Softmax en la última capa. Lo que hace es agarrar los valores brutos de salida y convertirlos en una distribución de probabilidad que suma exactamente 100 %.

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum e^{x_j}}$$

En la red multicapa usamos para calcular el error que propagábamos la función del Error Cuadrático Medio. Realmente usábamos su derivada justo en la línea `error = y - predicted_output`. Si vemos la función del error cuadrático medio o  $E$  entendemos que su derivada sea la indicada en el código. El  $\frac{1}{2}$  se añade por pura conveniencia matemática para que se cancele al derivar:

$$E = \frac{1}{2}(y - \hat{y})^2 \quad \frac{\partial E}{\partial \hat{y}} = -(y - \hat{y})$$

En esta nueva red para reconocer patrones usaremos una nueva función para calcular el error llamada **Cross-Entropy**. Su derivada es mucho más agresiva. Si la red está muy segura de que un número es un “3” pero en realidad es un “8”, la Cross-Entropy genera un gradiente gigantesco para obligar a la red a cambiar rápido.