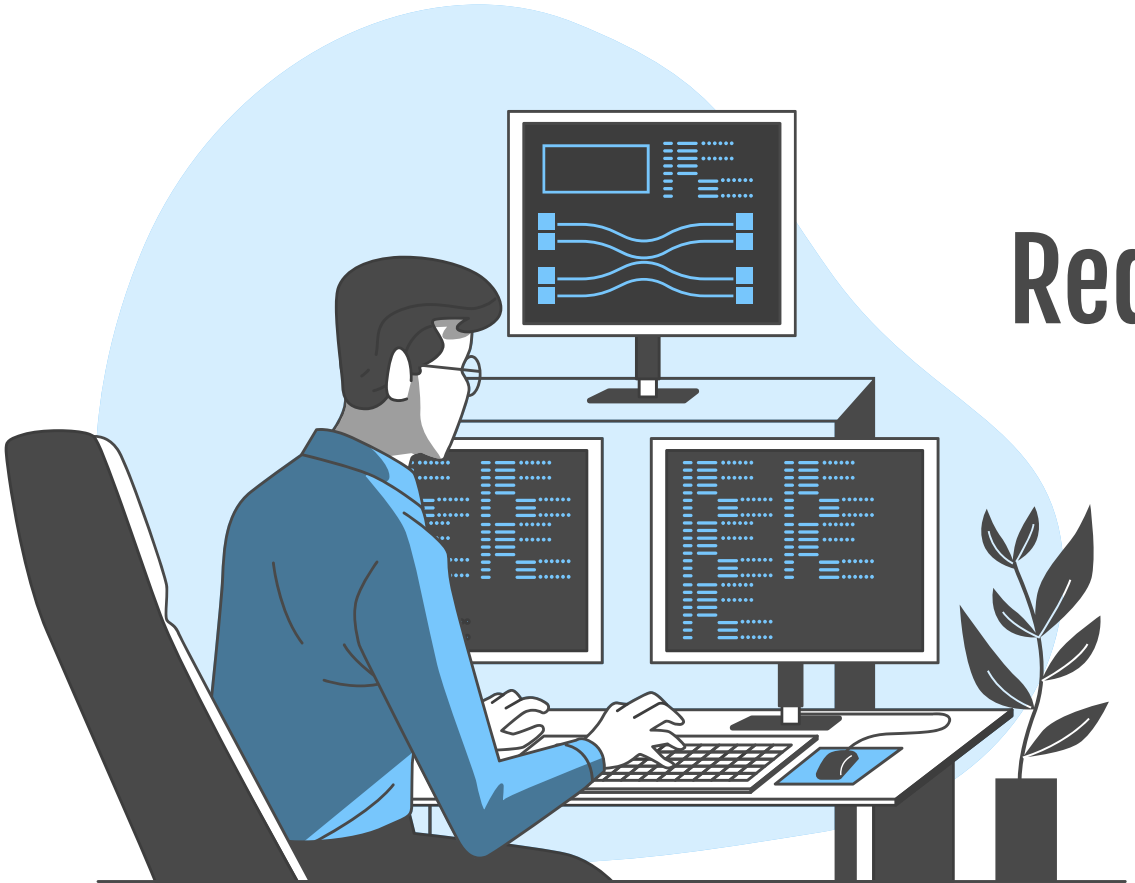
IPAR
ESTUDIOS



# Redes Neuronales

# ¿Qué es el Deep Learning?

Las redes neuronales artificiales son modelos computacionales de Machine Learning que están constituidos por unidades elementales de procesamiento, denominadas nodos o neuronas artificiales, dispuestas en capas y con muchas conexiones entre ellas. Estos sistemas aprenden de la experiencia (datos) guardando el conocimiento en la configuración de los valores de ciertos parámetros que los caracterizan, los pesos sinápticos.

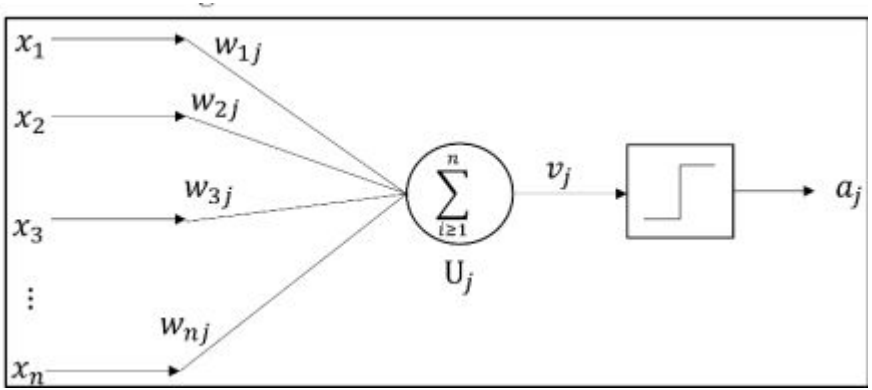
# ¿Qué es el perceptron?

El **perceptrón** es el **modelo más sencillo de red neuronal** y fue propuesto por Frank Rosenblatt en 1957. Su importancia radica en que es un **clasificador lineal** entrenable por una regla de aprendizaje muy intuitiva.

# Perceptron

Se trata de una unidad de procesamiento que combina linealmente $n$ señales de entrada $xi$ multiplicando cada una de ellas por un peso sináptico $wij$ para producir una entrada neta $vj$

$$v_j = w_{1j}.x_1 + w_{2j}.x_2 + \cdots + w_{nj}.x_n$$



# Perceptron

Un perceptron está compuesto por:

1. **Entradas** $\mathbf{x} = (x_1, x_2, \ldots, x_d)$.
2. **Pesos** $\mathbf{w} = (w_1, w_2, \ldots, w_d)$.
3. **Sesgo** (bias) $b$.
4. **Función de activación escalón**:

$$f(z) = \begin{cases} +1, & \text{si } z \geq 0, \\ -1, & \text{si } z < 0. \end{cases}$$

donde $z = \mathbf{w} \cdot \mathbf{x} + b$.

# Perceptron

La librería scikit-learn nos permite importar el algoritmo fácilmente

```
from sklearn.linear_model import Perceptron

clf = Perceptron(max_iter=2000, random_state=40, n_jobs=-1)
clf.fit(X_train, y_train)
```

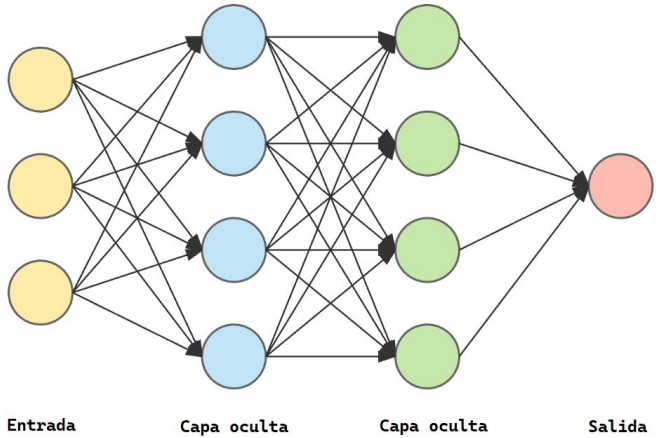
```
# Número de parámetros que forman el modelo
clf.coef_.shape
```

```
# Parámetros bias/intercept
clf.intercept_
```

# Del perceptrón a la Red Neuronal Multicapa (MLP)

## 2.1 Capas y neuronas

- **Capa de entrada**: no realiza cómputo, sólo pasa los valores $x$.
- **Capas ocultas**: una o varias; cada neurona ejecuta $z = Wx + b$ y aplica una activación no lineal.
- **Capa de salida**: produce una o más salidas $\hat{y}$ con la activación apropiada para la tarea.



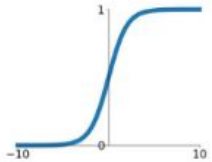
# Funciones de activación

- **Las funciones de activación son las funciones que sirven para activar las neuronas en las capas ocultas.**

## Activation Functions

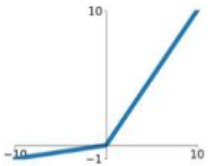
**Sigmoid**

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



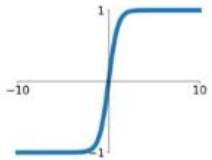

**Leaky ReLU**

$$\max(0.1x, x)$$




**tanh**

$$\tanh(x)$$

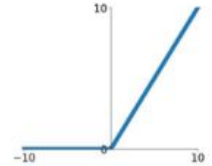



**Maxout**

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

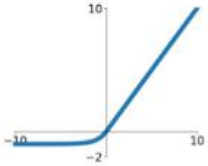
**ReLU**

$$\max(0, x)$$




**ELU**

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



# Funciones de activación

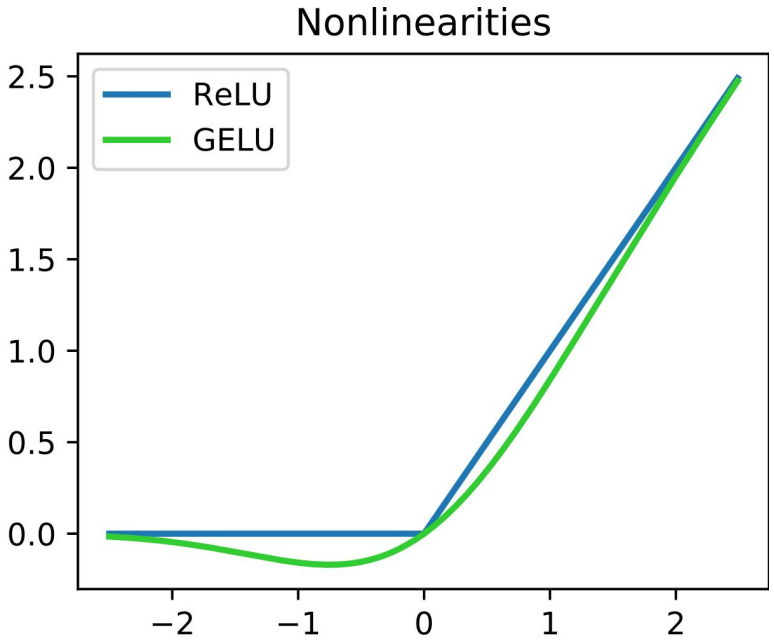
## 1. ReLU (Rectified Linear Unit)

$$f(x) = \max(0, x)$$

- **Uso:** capas ocultas
- **Ventajas:** rápida, simple, evita saturación (cuando no se aplana como la sigmoide)
- **Problemas:** neuronas pueden "morir" (si siempre reciben valores negativos)

# Funciones de activación

## 1. ReLU (Rectified Linear Unit)



# Funciones de activación

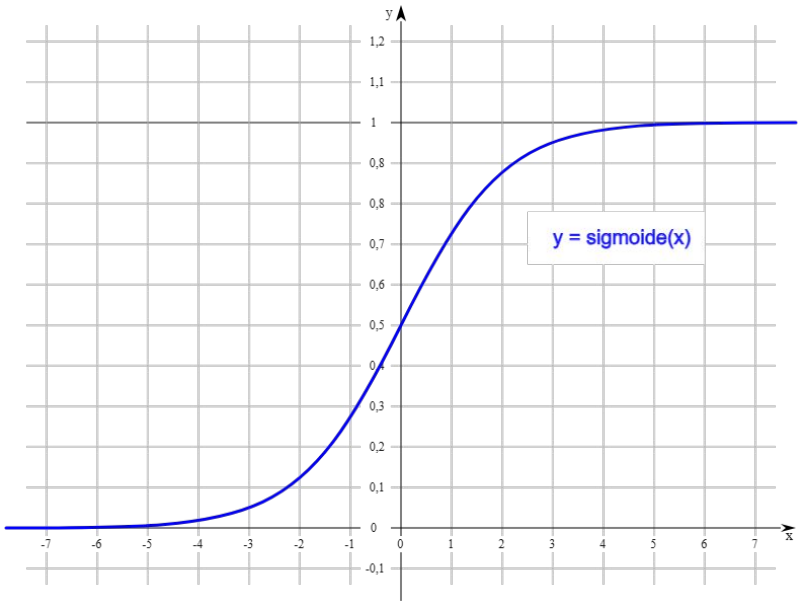
## 2. Sigmoide (Logística)

$$f(x) = \frac{1}{1 + e^{-x}}$$

- **Uso:** salidas binarias (clasificación)
- **Ventajas:** salida entre 0 y 1
- **Problemas:** gradiente pequeño para valores extremos (desvanecimiento del gradiente)

# Funciones de activación

## 2. Sigmoide (Logística)



## 3. Funciones de activación

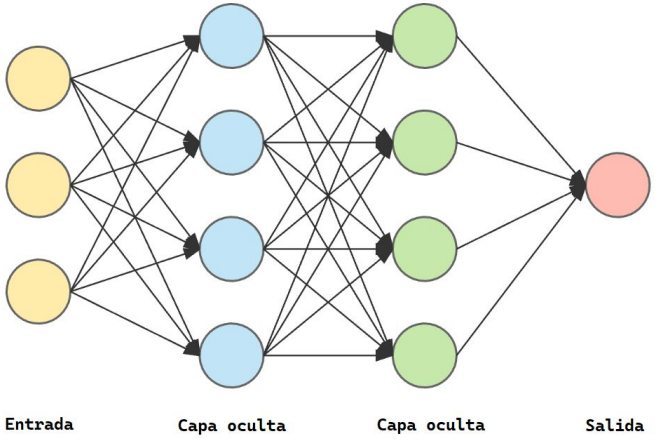
| Nombre | Fórmula | Rango | Uso típico |
|---|---|---|---|
| **ReLU** | $f(x) = \max(0, x)$ | $[0, \infty)$ | Capas ocultas de casi todas las CNN/MLP modernas. |
| **Sigmoide** | $\sigma(x) = \frac{1}{1 + e^{-x}}$ | $(0, 1)$ | Salida en clasificación binaria o capas ocultas antiguas. |
| **Tanh** | $\tanh(x)$ | $(-1, 1)$ | Oculta cuando se quiere salida centrada en 0. |
| **Softmax** | $\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$ | Probabilidad (suman 1) | Salida en clasificación multiclase. |
| **Lineal** | $f(x) = x$ | $(-\infty, \infty)$ | Salida en regresión. |

# Funciones de activación en la salida

1. **Sigmoide (Logística) para Clasificación binaria**
2. **Softmax para Clasificación multiclase**
3. **Regresión lineal**

# Propagación hacia adelante (Forward Pass)

La propagación hacia adelante (o "forward propagation") es el proceso por el cual una red neuronal procesa datos de entrada para generar una salida o predicción. En esencia, es el "cálculo" que realiza la red para transformar los datos de entrada en una salida final.



# Propagación hacia adelante (Forward Pass)

1. **Multiplicación**: producto punto (o matriz) en cada capa.
2. **Sesgo**: se suma a cada neurona.
3. **Activación**: se aplica $f$ neurona a neurona.
4. **Salida final**: se obtiene $\hat{y}$ tras la última capa.

Ejemplo con una capa oculta de 2 neuronas y ReLU:

$$\mathbf{z}^{(1)} = \mathbf{x}\,\mathbf{W}^{(1)} + \mathbf{b}^{(1)}$$
$$\mathbf{a}^{(1)} = \mathrm{ReLU}(\mathbf{z}^{(1)})$$
$$z^{(2)} = \mathbf{a}^{(1)} \cdot \mathbf{W}^{(2)} + b^{(2)}$$
$$\hat{y} = z^{(2)} \quad \text{(si la salida es lineal)}$$

# Propagación hacia adelante (Forward Pass)

1. **Multiplicación**: producto punto (o matriz) en cada capa.
2. **Sesgo**: se suma a cada neurona.
3. **Activación**: se aplica $f$ neurona a neurona.
4. **Salida final**: se obtiene $\hat{y}$ tras la última capa.

Ejemplo con una capa oculta de 2 neuronas y ReLU:

$$\mathbf{z}^{(1)} = \mathbf{x}\,\mathbf{W}^{(1)} + \mathbf{b}^{(1)}$$
$$\mathbf{a}^{(1)} = \mathrm{ReLU}(\mathbf{z}^{(1)})$$
$$z^{(2)} = \mathbf{a}^{(1)} \cdot \mathbf{W}^{(2)} + b^{(2)}$$
$$\hat{y} = z^{(2)} \quad \text{(si la salida es lineal)}$$

# Funciones de Pérdida

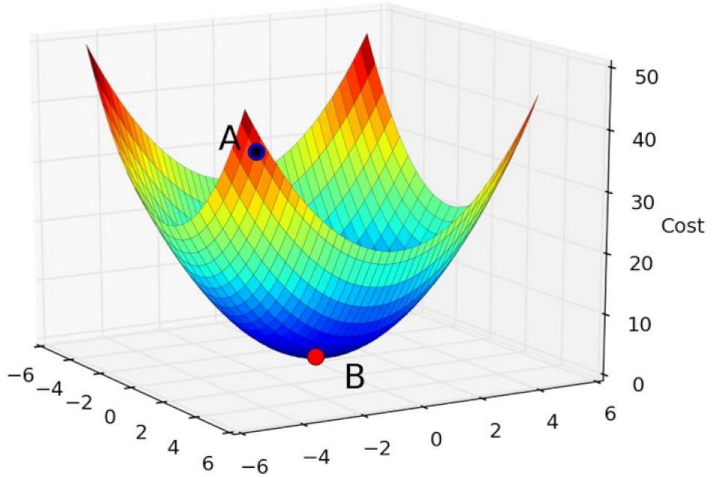
| Tipo de problema | Pérdida habitual | Fórmula resumida |
| --- | --- | --- |
| Regresión | **MSE** (Error Cuadrático Medio) | $L = \frac{1}{2}(y - \hat{y})^2$ |
| Clasificación binaria | **Binary Cross-Entropy** | $-\left[y \log \hat{y} + (1 - y) \log(1 - \hat{y})\right]$ |
| Clasificación multiclase | **Categorical Cross-Entropy** | $-\sum_k y_k \log \hat{y}_k$ |

# Funciones de Pérdida

Las funciones de pérdida en las redes neuronales son medidas que cuantifican la diferencia entre la salida predicha por el modelo y la salida real o deseada. En esencia, nos dicen qué tan bien está aprendiendo la red neuronal y cuánto se está equivocando en sus predicciones. Minimizar la función de pérdida durante el entrenamiento es clave para mejorar el rendimiento de la red.

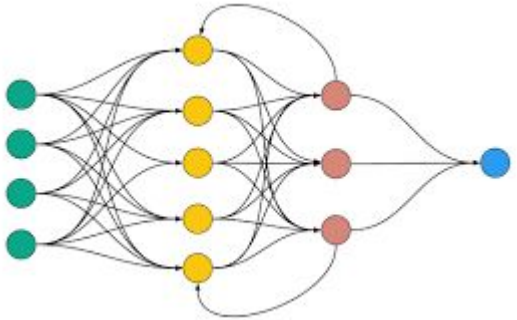
# Descenso de gradiente y retropropagación (Backpropagation)

El descenso de gradiente es un algoritmo de optimización iterativo que se utiliza para encontrar los mínimos de una función.



# Descenso de gradiente y retropropagación (Backpropagation)

La retropropagación en redes neuronales (también conocida como "backpropagation") es un algoritmo fundamental para entrenar redes neuronales artificiales, especialmente redes multicapa. Sirve para ajustar los pesos de las conexiones entre las neuronas con el objetivo de minimizar el error de predicción de la red.



# Descenso de gradiente y retropropagación (Backpropagation)

1. **Objetivo**: minimizar $L$ ajustando pesos y sesgos.
2. **Gradiente**: vector de derivadas $\frac{\partial L}{\partial w}$.
3. **Actualización básica**:

$$w \leftarrow w - \eta \frac{\partial L}{\partial w}$$

# Hiperparámetros clave

- **Tasa de aprendizaje** $\eta$: paso del descenso de gradiente.
- **Número de épocas**: veces que se recorre todo el conjunto de entrenamiento.
- **Tamaño de minibatch**: ejemplos usados por actualización.
- **Número de capas / neuronas**: capacidad del modelo.
- **Función de activación y pérdida**: dependen de la tarea.

# Resumen rápido para los ejercicios

1. **Producto punto** → suma ponderada $z$.
2. **Activación** → ReLU, sigmoide, etc.
3. **Forward pass** → repetir capa por capa.
4. **Loss** → MSE (si se pide).
5. **Gradiente** (para ejercicios de backprop.)
    - Error $e = y - \hat{y}$.
    - Derivada de la pérdida: MSE ⇒ $-e$.
    - Gradiente peso $w_i$: $-x_i\, e$.
6. **Actualizar**: $w_i \leftarrow w_i - \eta(-x_i e)$.

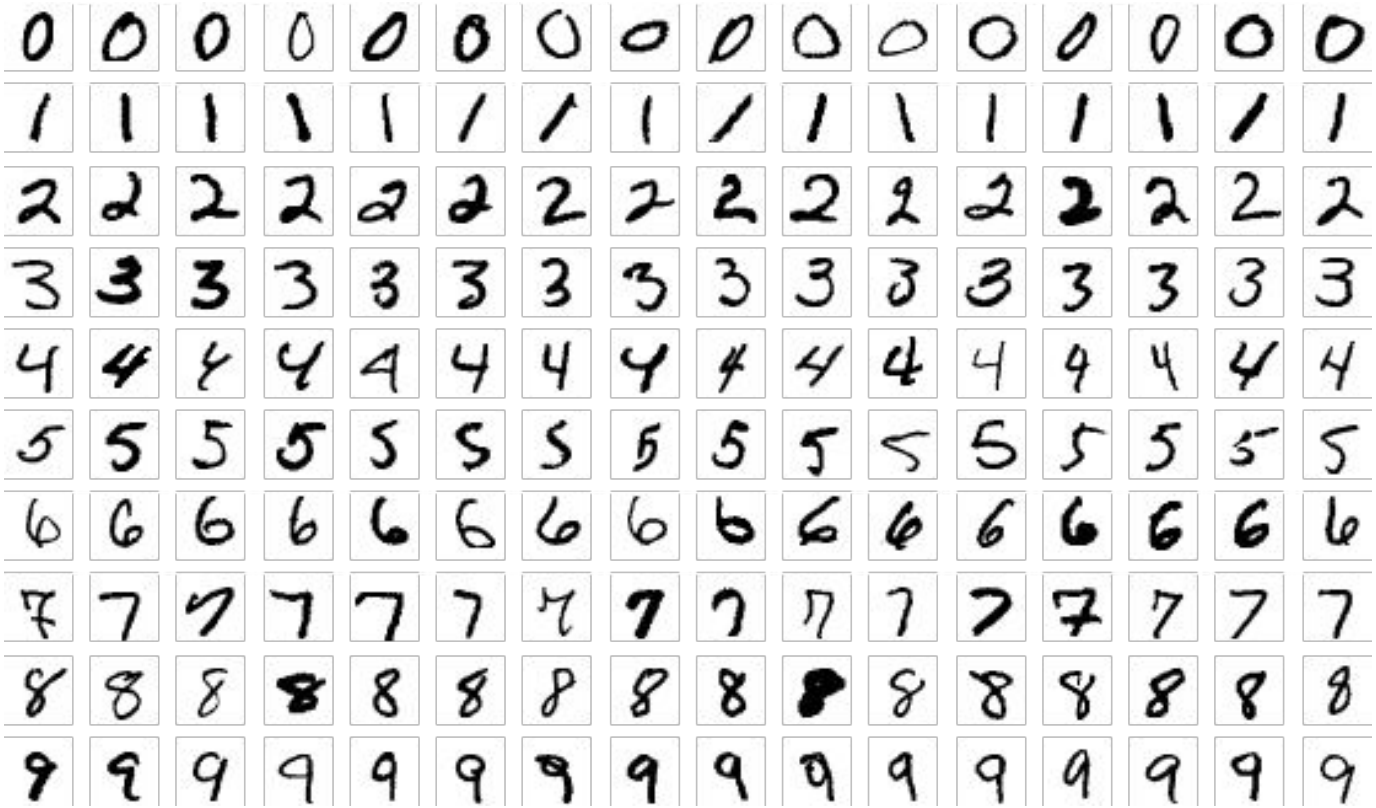
# ¿Qué es MNIST?

**MNIST** (Modified National Institute of Standards and Technology) es un conjunto de datos clásico usado para entrenar y evaluar algoritmos de aprendizaje automático, especialmente redes neuronales.

🔢 Contiene **imágenes de dígitos escritos a mano**, del 0 al 9.

- 60.000 imágenes de entrenamiento
- 10.000 imágenes de prueba
- Cada imagen tiene **28×28 píxeles**, en escala de grises
- Las etiquetas indican el dígito que representa (0 a 9)

# ¿Qué es MNIST?



# ¿Qué es MNIST?

```
# Importamos el conjunto de datos
from sklearn.datasets import fetch_openml

# Añadimos as_frame=False para forzar la devolución de un array
mnist = fetch_openml('mnist_784',  as_frame=False)
```

```
mnist.data
df = pd.DataFrame(mnist.data)
```

# PRACTICA

En este ejercicio práctico vamos a utilizar el Perceptrón para solucionar un problema más complejo. En este caso, nuestro objetivo será clasificar un conjunto de imágenes en base al número que se muestra en las mismas.

1. Lectura de Datos
2. Visualización de los datos
3. División del conjunto de datos
4. Entrenamiento del algoritmo
5. Predicción
6. Mostrar imagenes mal clasificadas