Esenciales de Deep Learning

Presentación intensiva de 2 horas

J. Peralta

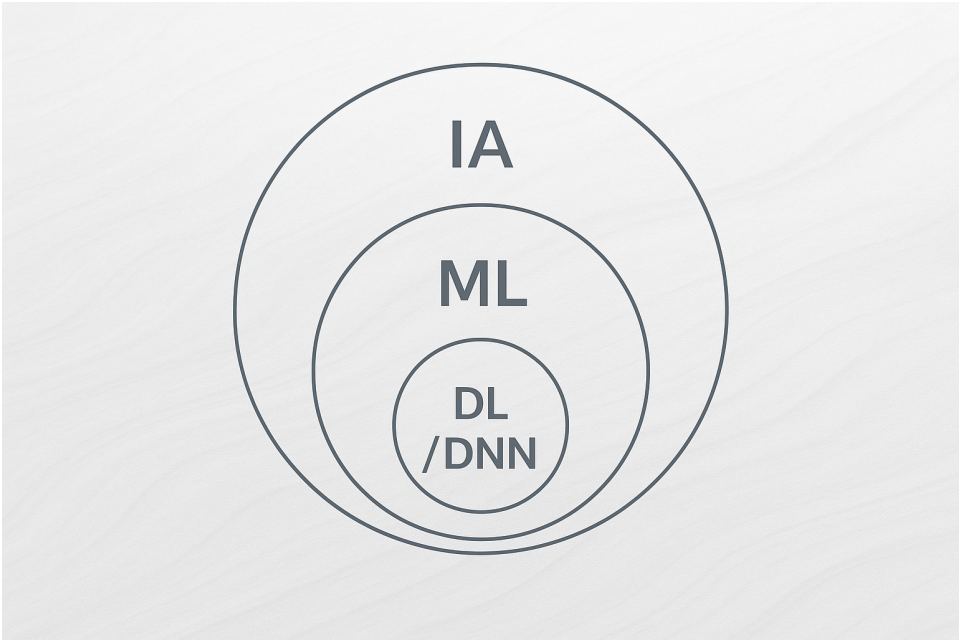
Universidad Andrés Bello

Julio 2025

Guía de la sesión

1. IA, ML y Deep Learning
2. Panorama general
3. Redes Neuronales
4. CNN en detalle
5. Regularización y Mejora de modelos
6. Ejemplos en Keras
7. Conclusiones

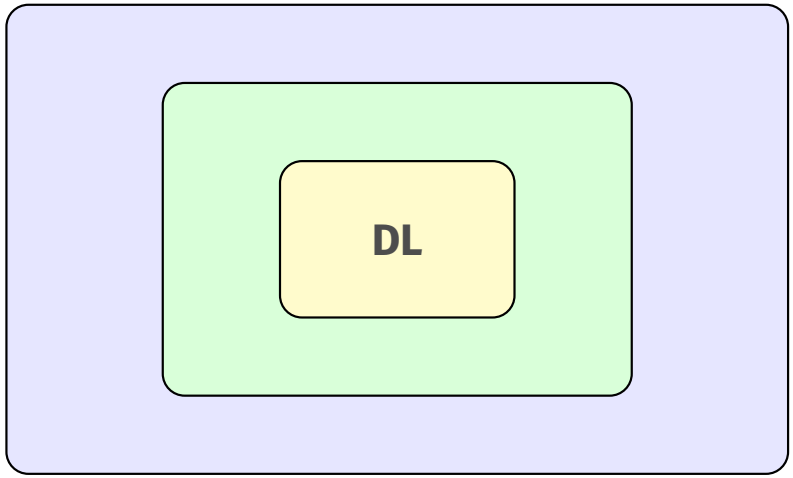
Mapa conceptual



- **IA** engloba cualquier técnica que emule comportamientos inteligentes.
- **ML** aprende patrones a partir de datos; es un subconjunto de IA.
- **DL / DNN** aprende representaciones jerárquicas con redes profundas; es un subconjunto de ML.

¿Qué entendemos por Inteligencia Artificial?

- **Tareas clásicas**:
  - Razonamiento → demostradores de teoremas (*Prover9*).
  - Planificación → rover *Sojourner* (NASA, 1997).
  - Percepción → detección de peatones en vehículos autónomos.
  - Procesamiento de lenguaje → ChatGPT, traducción automática.
- **Paradigmas históricos**:
  - **GOFAI** (reglas `IF–THEN`, lógica de predicados, MYCIN).
  - **Búsqueda** (A*, Minimax + αβ; *Deep Blue*, 1997).
  - **Probabilístico** (redes bayesianas, POMDP).
- **Éxitos tempranos**: diagnóstico médico, ajedrez, planificación de satélites.

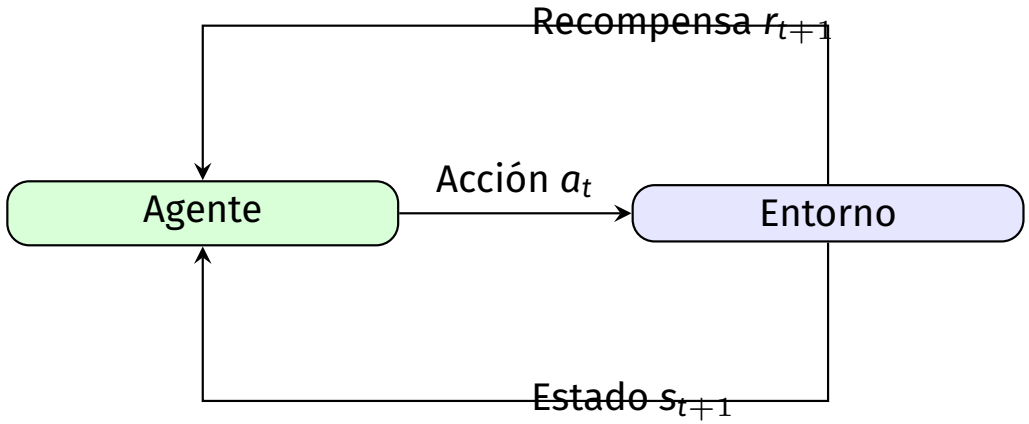


Clave

No toda IA *aprende*; muchos sistemas se basan en reglas codificadas a mano.

Machine Learning: aprender de los datos

- **Objetivo**: construir modelos que *generalicen* a partir de datos.
- **Paradigmas fundamentales**:
  - Supervisado: clasificación (*spam* / *no-spam*), regresión (predicción de demanda eléctrica).
  - No supervisado: k-means en catálogos galácticos, reducción de dimensionalidad (t-SNE en MNIST).
  - Por refuerzo: AlphaGo, control de drones, trading algorítmico.
- **Algoritmos clásicos**: Regresión lineal, k-NN, Árboles y Bosques Aleatorios, SVM, k-means.
- **Tendencias recientes**: AutoML, aprendizaje federado en dispositivos móviles.



Deep Learning y redes neuronales profundas

- Redes con muchas capas (+ millones de parámetros) que aprenden representaciones jerárquicas.
- Arquitecturas: CNN, RNN/LSTM, Transformers, GNN.
- Escalan bien con grandes volúmenes de datos y potencia GPU/TPU.

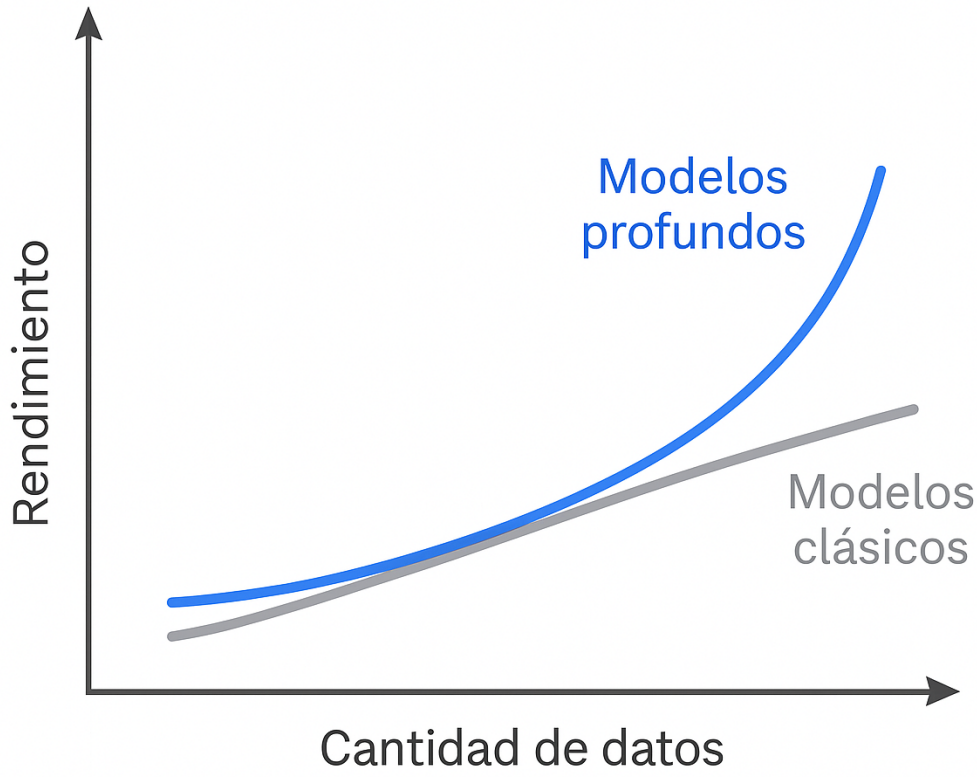
Observación

DL = representación + aprendizaje $\Rightarrow$ elimina ingeniería manual de características.

Comparativa IA vs ML vs DL

Aspecto	IA	ML	DL
Datos necesarios	Bajo–medio	Medio	Alto
Ingeniería manual	Alta	Media	Baja
Interpretabilidad	Alta	Media	Baja
Hardware	CPU	CPU/GPU	GPU/TPU
Tareas típicas	Planificación	Predicción	Visión, lenguaje

Rendimiento vs cantidad de datos



Regla empírica

Modelos profundos superan a los clásicos cuando los datos ↑ y la complejidad del problema es alta.

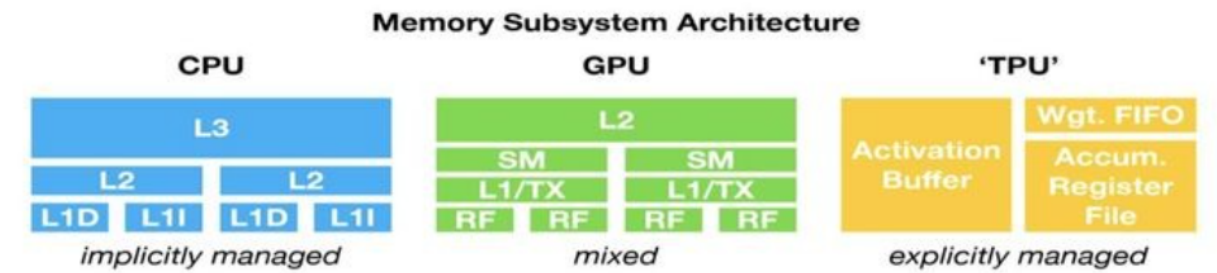
Hardware y escalabilidad

- Entrenamiento $\sim$ GFlops–TFlops; inferencia en dispositivos móviles posible.
- GPUs y TPUs aceleran multiplicaciones matriz–vector.
- Distribución de datos y modelos: Data Parallel & Model Parallel.

CPU vs GPU vs TPU

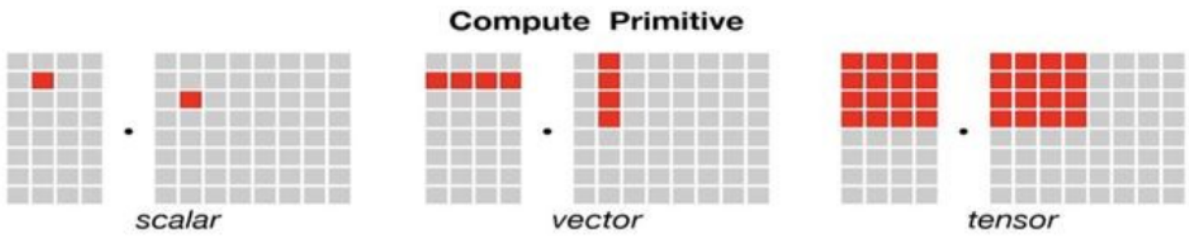
Memory Subsystem Architecture

This image captures the memory architecture of CPU, GPU and TPU:



Compute Primitive

This image summarizes the compute primitive (smallest unit) in CPU, GPU and TPU:



Aplicaciones destacadas

IA (general)

- Sistemas expertos médicos.
- Planificación automática en logística.
- Robótica industrial.

ML clásico

- Créditos bancarios.
- Motores de recomendación básicos.
- Detección de fraude (reglas + SVM).

Deep Learning

- Reconocimiento de imágenes (ImageNet).
- Traducción automática (Transformers).
- Chatbots y LLMs (GPT-x).

Conclusiones

- IA $\supset$ ML $\supset$ DL/DNN.
- DL domina cuando hay datos masivos y poder de cómputo.
- Algoritmos clásicos siguen siendo competitivos con pocos datos o requisitos de interpretabilidad altos.

Enlaces sugeridos

- Yann LeCun, “Deep Learning”, http://yann.lecun.com/deep-learning/
- MIT 6.S191, https://introtodeeplearning.mit.edu

Deep Learning en el contexto de la IA

- **Inteligencia Artificial (IA):** Sistemas capaces de realizar tareas que normalmente requieren inteligencia humana.
- **Machine Learning (ML):** Sub–campo de IA que aprende patrones a partir de datos (*programación basada en datos*).
- **Deep Learning (DL):** Sub–campo de ML que utiliza redes neuronales profundas para modelar funciones altamente no–lineales.

Regla de oro

Mientras más datos, mejor se comporta el modelo *(a diferencia de muchos algoritmos clásicos).*

Taxonomía rápida de ML

Aprendizaje supervisado

- Clasificación
- Regresión
- Ej.: filtro de spam, predicción de precios.

Aprendizaje por refuerzo

- Agente + entorno + recompensas.
- Ej.: AlphaGo, control de robots.

Aprendizaje no supervisado

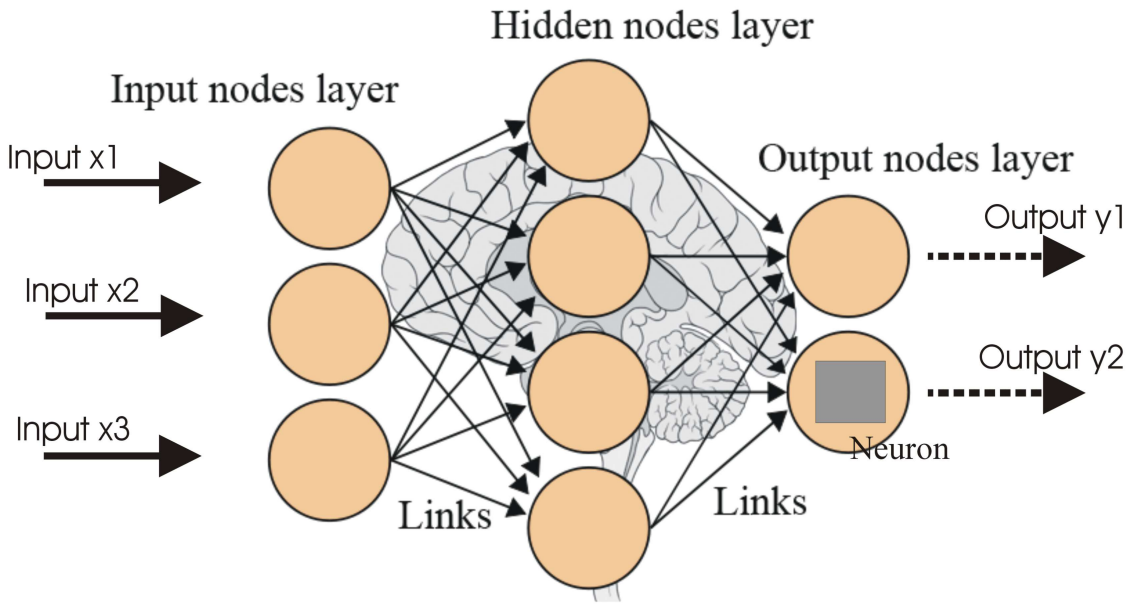
- Clustering (k-means)
- Modelos generativos (GAN, autoencoders)

Semi–supervisado / Auto–supervisado

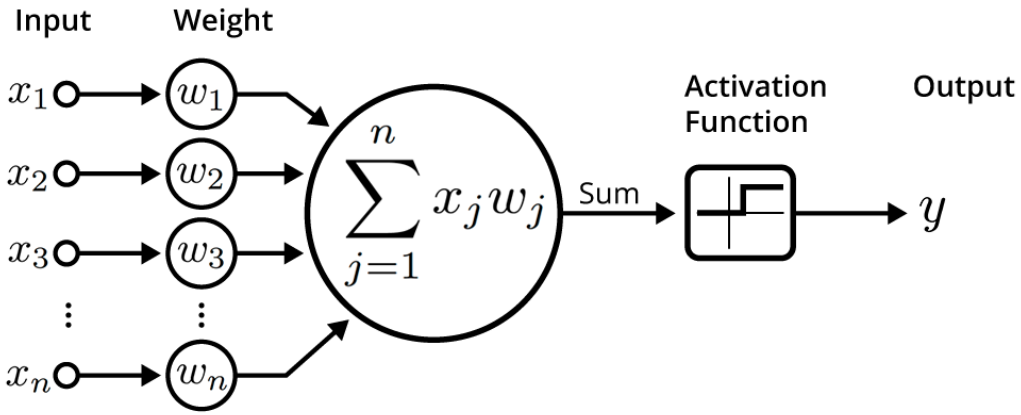
- Mezcla etiquetas + datos sin etiquetar.

Descomponiendo DL

- En su forma básica los modelos DL son diseñados usando redes neuronales.
- Una red es una organización jerárquica de neuronas con conección a otras neuronas.
- Las neuronas pasan un mensaje o señal a otras neuronas basadas en lo que reciben, y así aprenden con algún mecanismo de *feedback*
- A la derecha hay una representación simplista de una red neuronal.



1/5 - La neurona artificial: idea básica

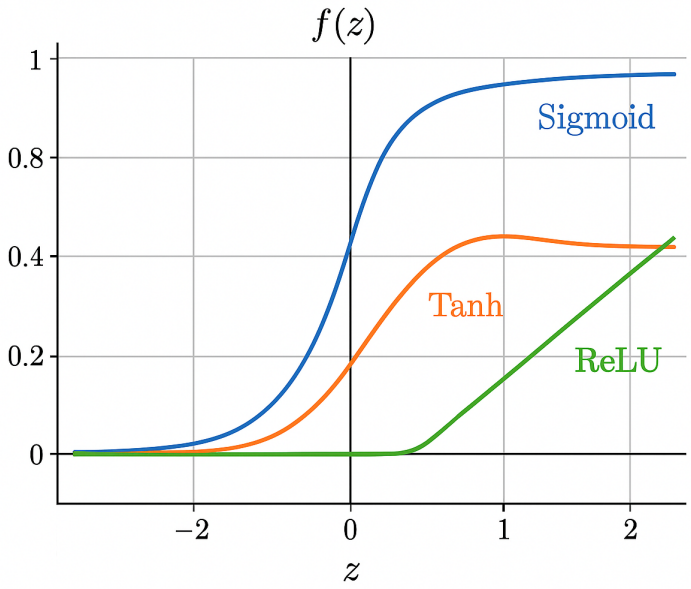


An illustration of an artificial neuron. Source: Becoming Human.

$$y = f\Big(\sum_{i=1}^{n} x_i\,\omega_i + b\Big)$$

- **Potencial**: suma ponderada + sesgo.
- **Activación** f: introduce no-linealidad.
- Con varias neuronas $\rightarrow$ capa densa.

- Sigmoide: $\sigma(z) = 1/(1 + e^{-z})$
- TANH: rango $[-1, 1]$
- **ReLU**: $\max(0, z)$
- ReLU6, Leaky-ReLU, GELU ...



Tip

ReLU acelera el entrenamiento profundo y evita saturación de gradientes.

3/5 - Keras: construyendo una neurona

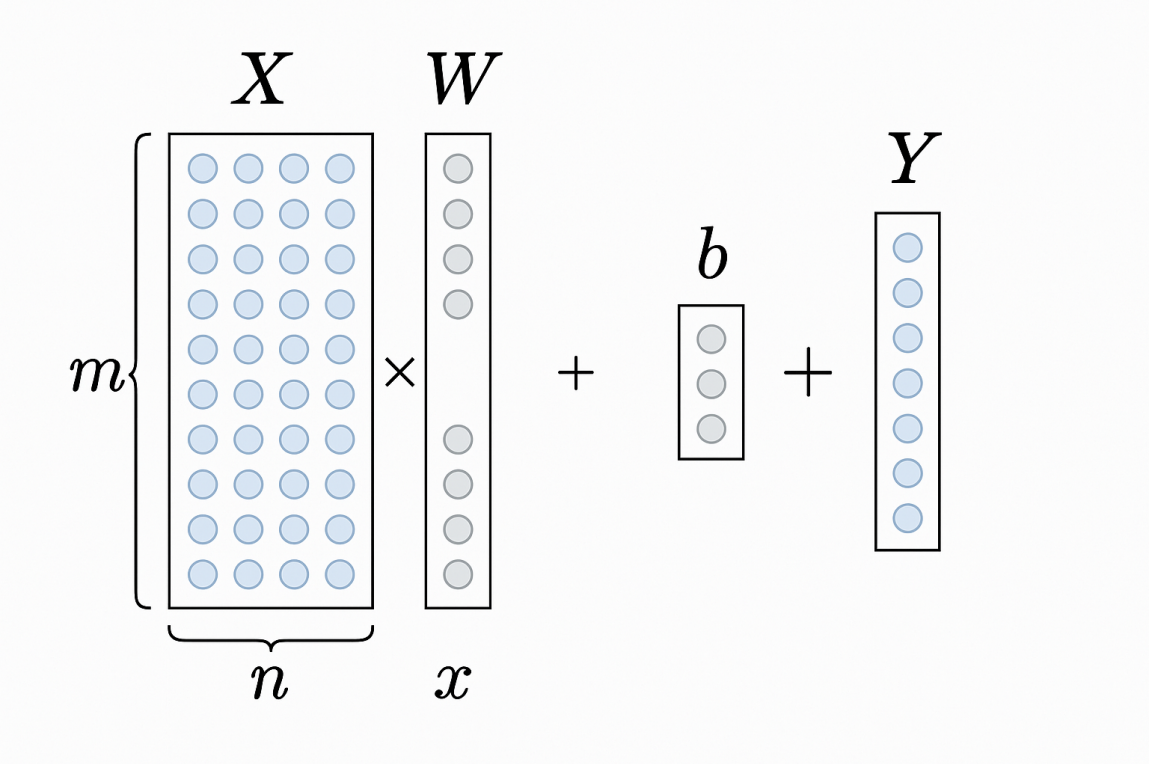
```python
from tensorflow.keras import layers, Sequential

# x -> y = ReLU(Wx + b)
model = Sequential([
    layers.Dense(1, activation='relu',
                 input_shape=(n_features,))
])
y_pred = model(x_batch)
```

- 'Dense(1)' = 1 neurona.
- Parámetros aprendibles: $n_{\text{feat}} + 1$.

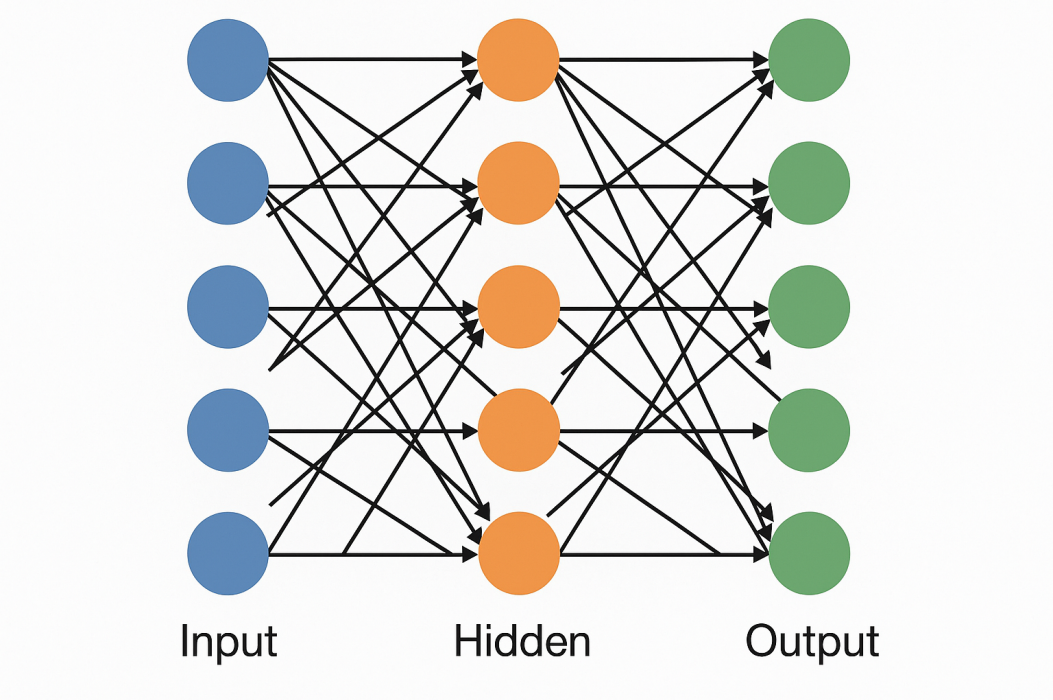
4/5 - Vectorización y batch-processing

- Entrenamos m ejemplos en paralelo: $Y = f(XW + b)$
- 'X' $(m \times n)$ multiplica los pesos $(n \times 1)$.
- GPU = miles de neuronas / ejemplos simultáneos.



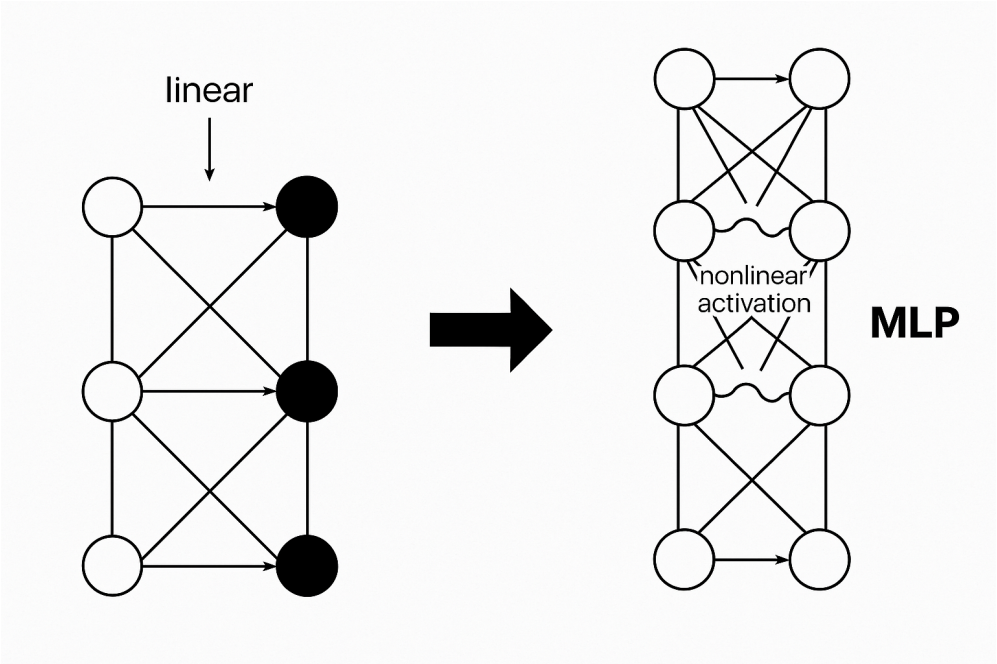
5/5 - Capas densas encadenadas

- Salida de una capa = entrada de la siguiente.
- Cada capa aprende representaciones de mayor nivel.
- Profundidad $\uparrow \Rightarrow$ capacidad $\uparrow$, riesgo de sobreajuste $\uparrow$.



De la *idea* de encadenar capas a la *arquitectura* Perceptrón Multicapa

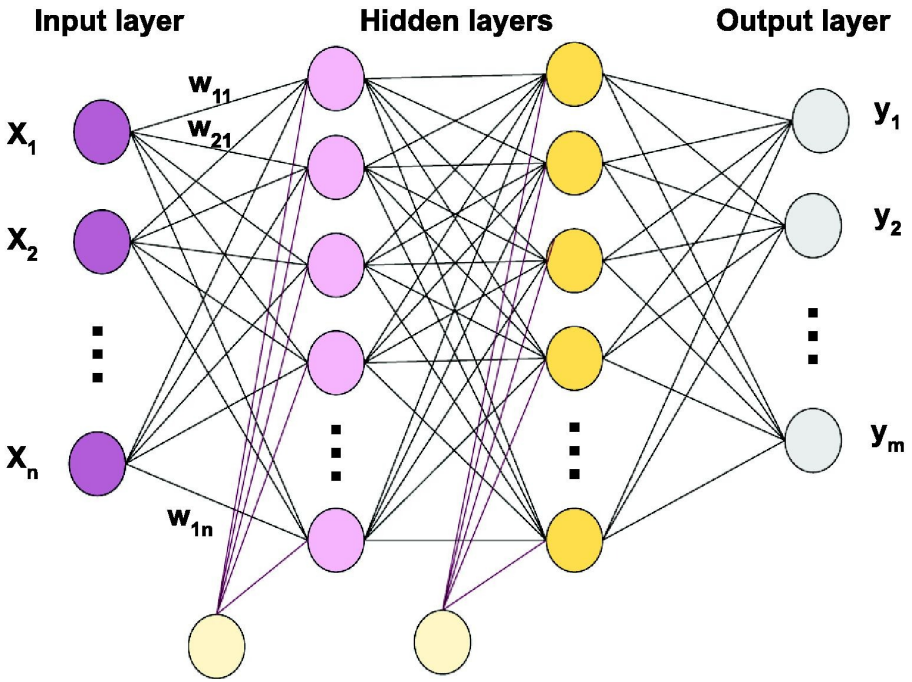
- Una red de **capas densas encadenadas** (*feed–forward*) ya es capaz de aproximar funciones complejas, pero sigue siendo lineal si no aplicamos activaciones.
- Al intercalar **activaciones no lineales** (ReLU, tanh, GELU, etc.) cada capa aprende representaciones más abstractas.
- El siguiente paso natural es el **Perceptrón Multicapa (MLP)**, que extiende esta idea a varias capas ocultas y se convierte en un *aproximador universal*.
- A continuación exploraremos *arquitecturas clásicas*: MLP, CNN, RNN, etc.



1/5 - Perceptrón multicapa (MLP)

- Capas densas + activaciones no-lineales.
- Adecuado para datos tabulares o embeddings.

```
model = Sequential([
    layers.Dense(128, activation='relu', input_shape=(784,)),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```



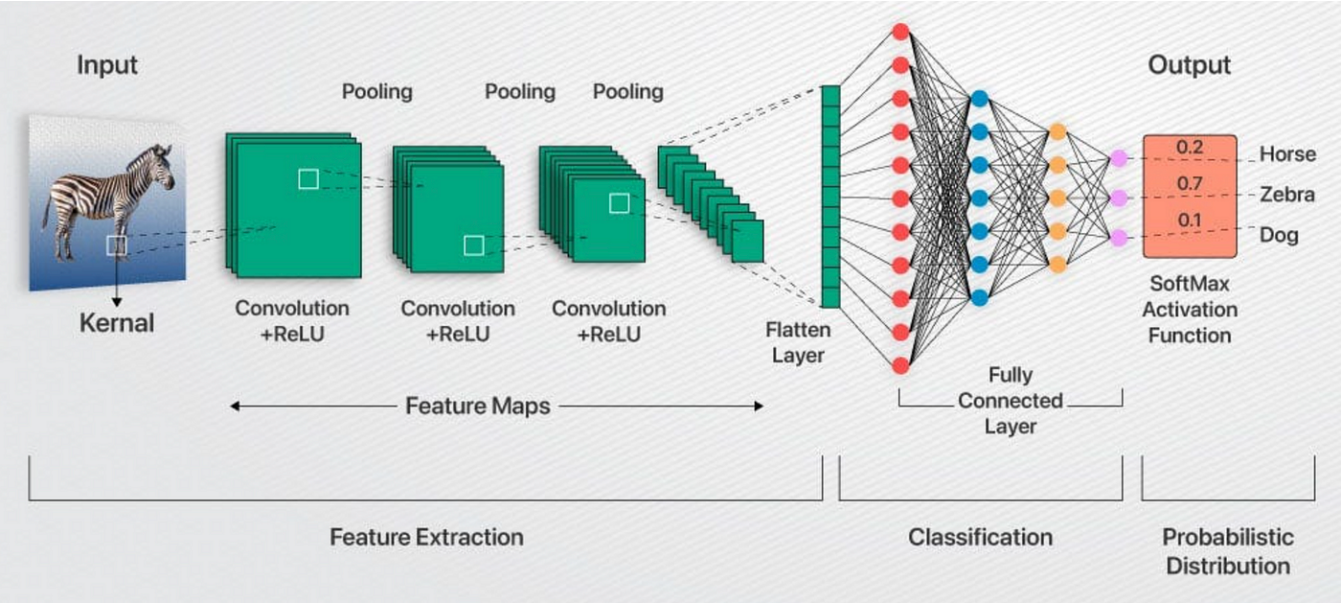
[1]

[1] https://doi.org/10.1016/j.neucom.2023.126327

2/5 - Convolutional Neural Networks (CNN)

- Filtros que comparten pesos $\rightarrow$ detección local.
- Pooling reduce dimensionalidad y traslación.

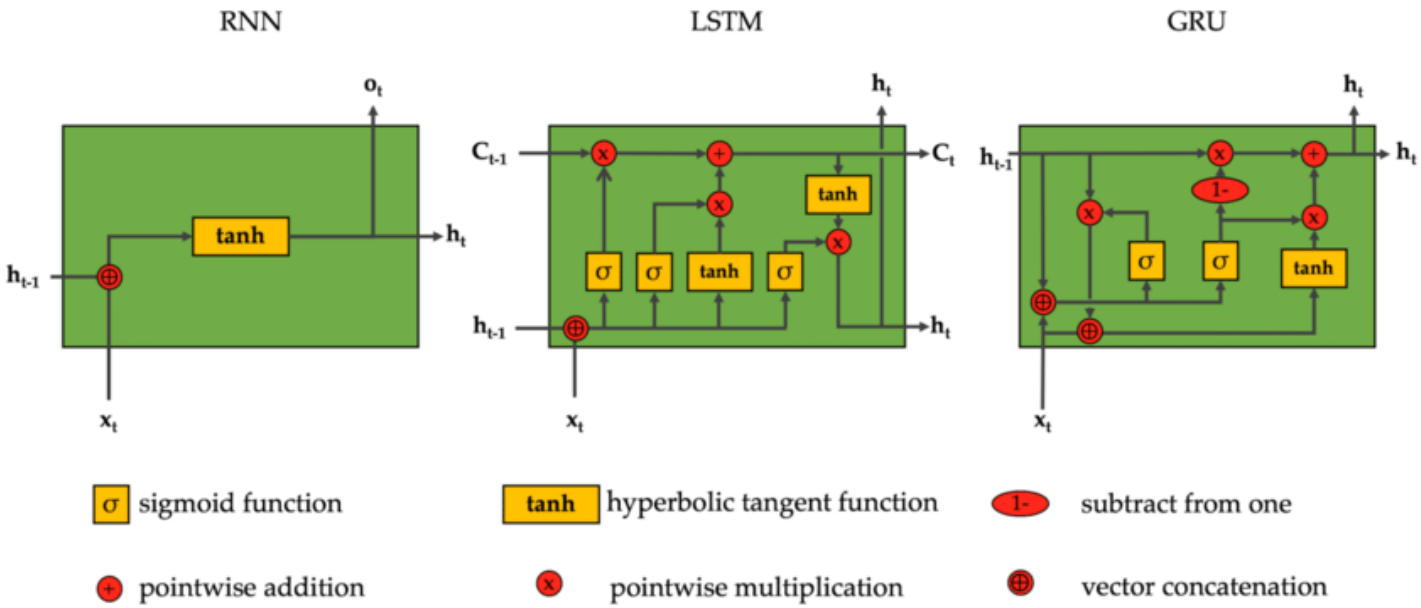
```
model = Sequential([
  layers.Conv2D(32, 3, activation='relu',
                input_shape=(32,32,3)),
  layers.MaxPooling2D(2),
  layers.Conv2D(64, 3, activation='relu'),
  layers.Flatten(),
  layers.Dense(10, activation='softmax')
])
```



3/5 - Redes recurrentes (RNN, LSTM, GRU)

- Estado interno h_t que se recicla en el tiempo.
- Modela secuencias: texto, audio, series temporales.

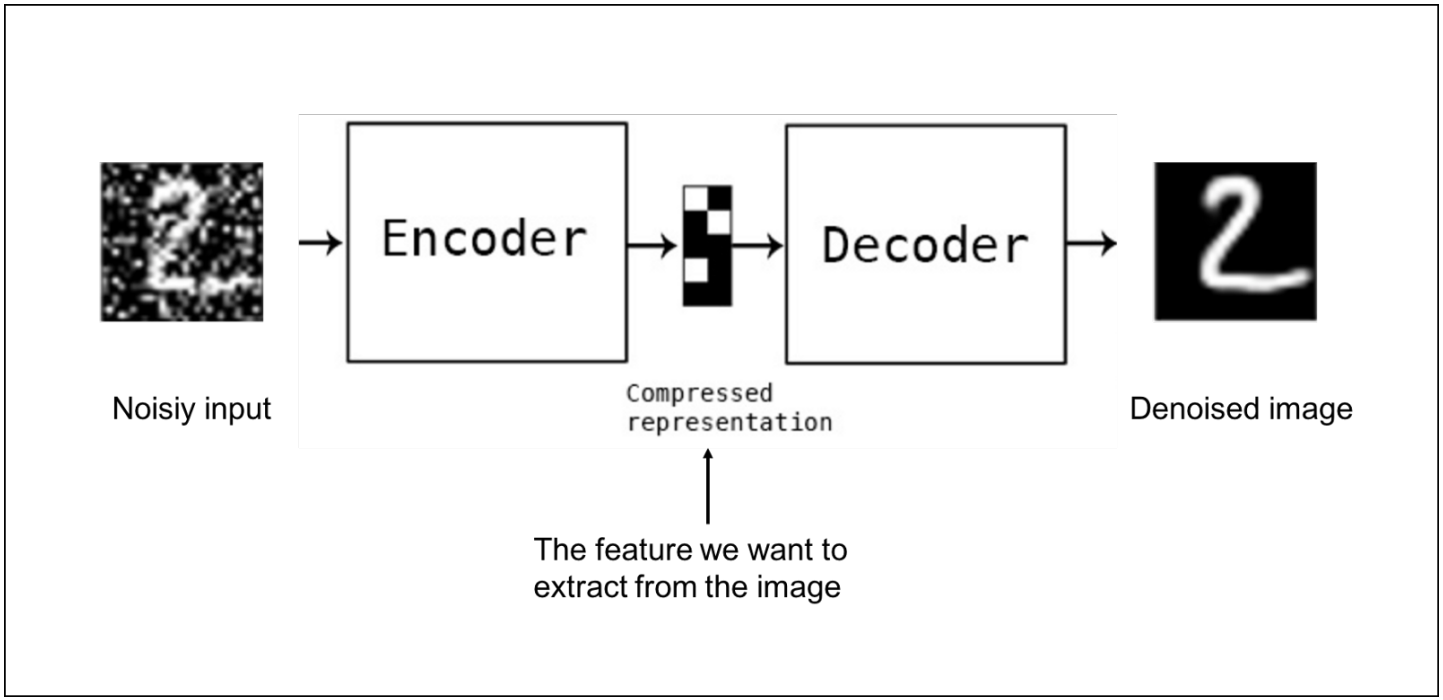
```
model = Sequential([
    layers.Embedding(vocab, 128),
    layers.LSTM(64),
    layers.Dense(1, activation='sigmoid')
])
```



4/5 - Autoencoders

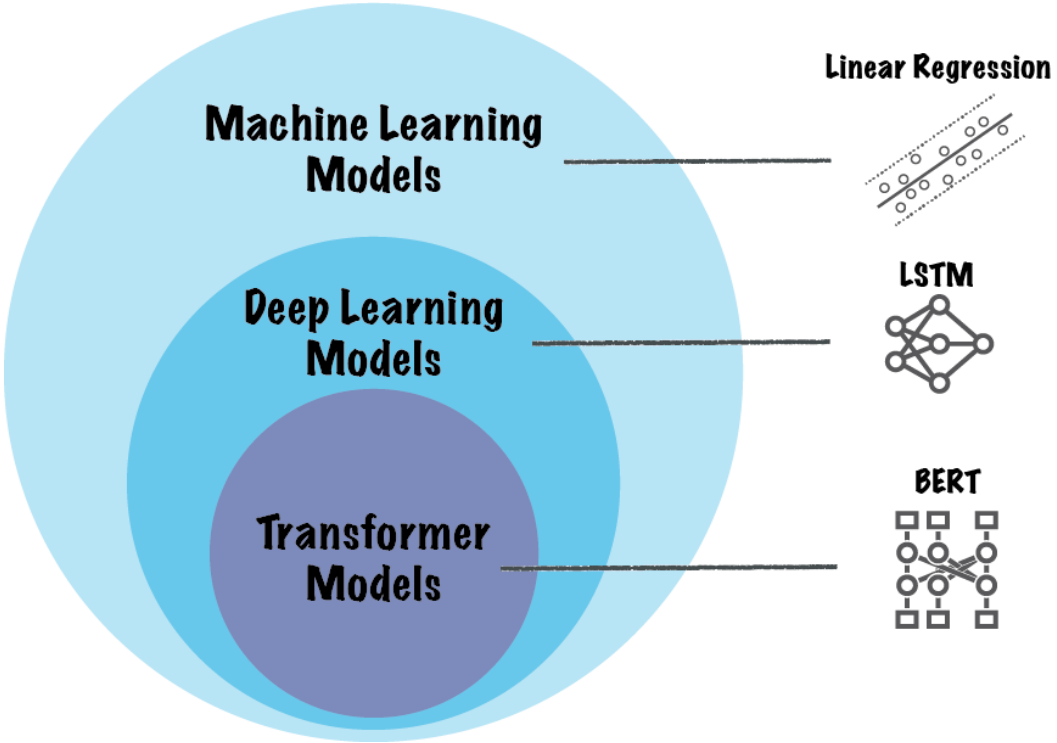
- Codificador $x \to z$; decodificador $z \to \hat{x}$.
- Objetivo: reconstruir entrada $\Rightarrow$ compresión no lineal.
- Base de denoising, GAN, VAEs.

```
encoded = layers.Dense(32, activation='relu')(inputs)
decoded = layers.Dense(784, activation='sigmoid')(encoded)
autoencoder = keras.Model(inputs, decoded)
```

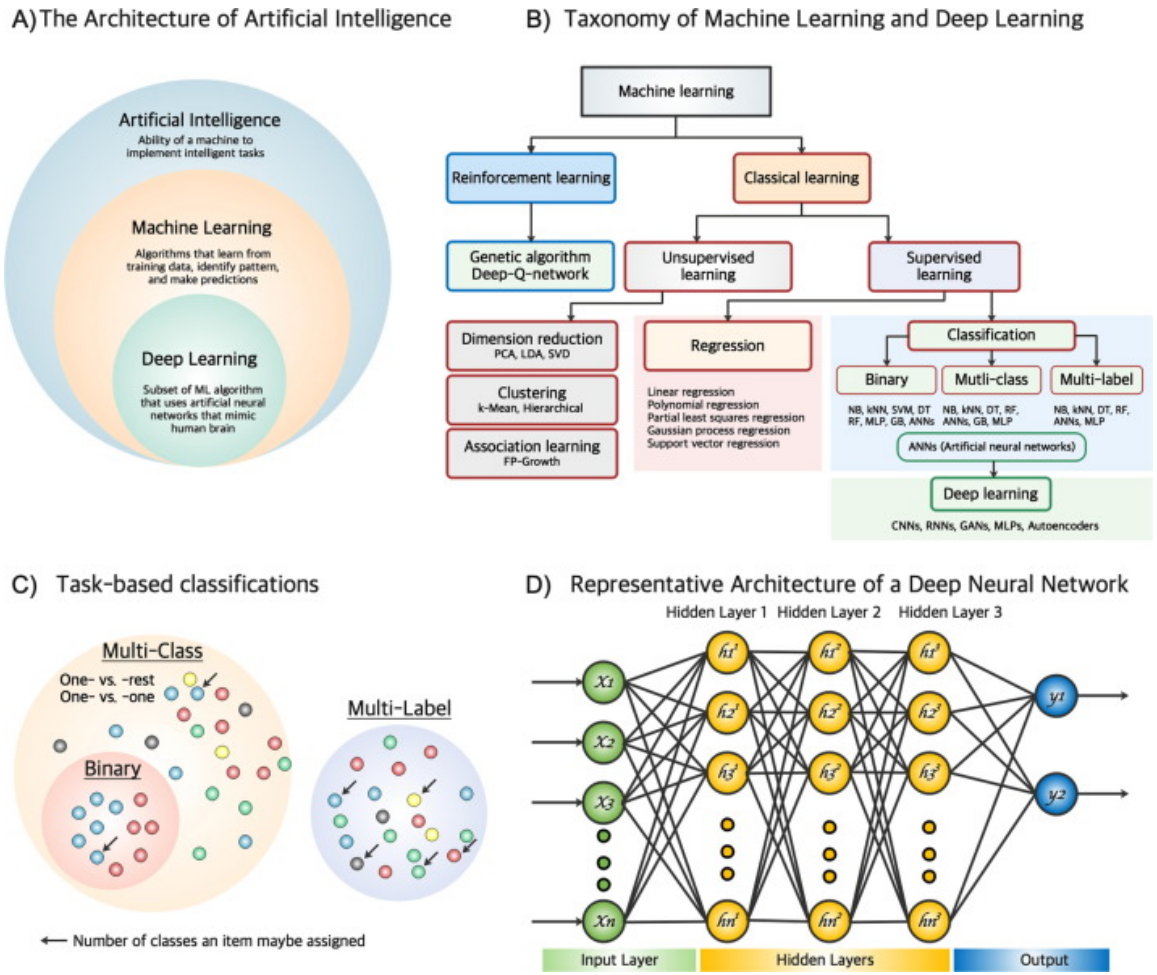


5/5 - Transformers (bonus)

- Mecanismo de **atención**: pesos dinámicos por token.
- Paraleliza secuencias mejor que RNN.
- BERT, GPT-x, Vision-Transformer.



1/5 - Pipeline de entrenamiento



1. Forward pass: $x \rightarrow y_{pred}$
2. Cálculo de pérdida J
3. Back-prop: gradientes $\nabla_\omega J$
4. Actualización de pesos

Clasificación multiclase

$$\mathcal{L}_{\text{CE}} = -\sum_{k=1}^{C} y_k \, \log \hat{y}_k$$

(Sólo usa la fila verdadera del vector one–hot, se implementa como `CategoricalCrossentropy`).

Regresión

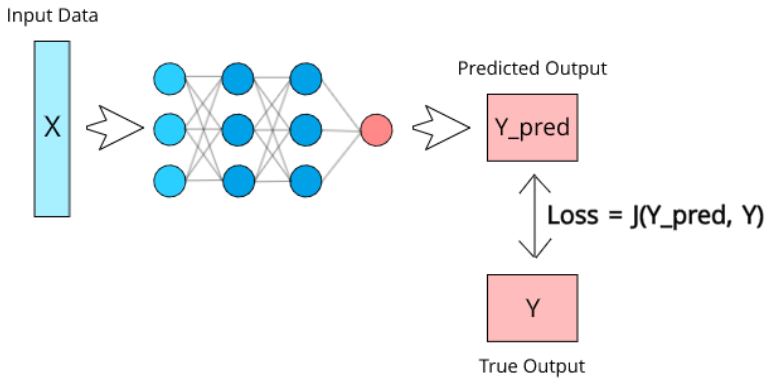
$$\mathcal{L}_{\text{MSE}} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Penaliza errores grandes de forma cuadrática.

Autoencoder variacional (VAE)

$$\mathcal{L}_{\text{ELBO}} = \underbrace{\mathbb{E}_{q_\phi(z|x)}[-\log p_\theta(x|z)]}_{\text{reconstrucción}} + \underbrace{\text{KL}(q_\phi(z|x) \parallel p(z))}_{\text{regularización}}$$

Equilibra fidelidad y regularización bayesiana.



Función de pérdida.

Descenso de gradiente estocástico (SGD)

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}(\theta_t; \mathcal{B}_t)$$

- Ligero, sin memoria extra.
- Ruido $\Rightarrow$ ayuda a escapar de mínimos locales.
- Sensible a la elección de η y al *schedule*.

Adam (Adaptive Moment Estimation)

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

$$\hat{m}_t = m_t / (1-\beta_1^t)$$

$$\hat{v}_t = v_t / (1-\beta_2^t)$$

$$\theta_{t+1} = \theta_t - \eta\, \hat{m}_t / \left(\sqrt{\hat{v}_t} + \varepsilon\right)$$

- Paso adaptativo por parámetro.
- Converge rápido, estable en datos dispersos.
- Más memoria ($2\times$ parámetros) $\&$ requiere $\beta_{1,2}$ y ε.

Otros optimizadores: **RMSProp**, **Adagrad**, **AdamW** (decay), **Nadam** (momentum de Nesterov)...

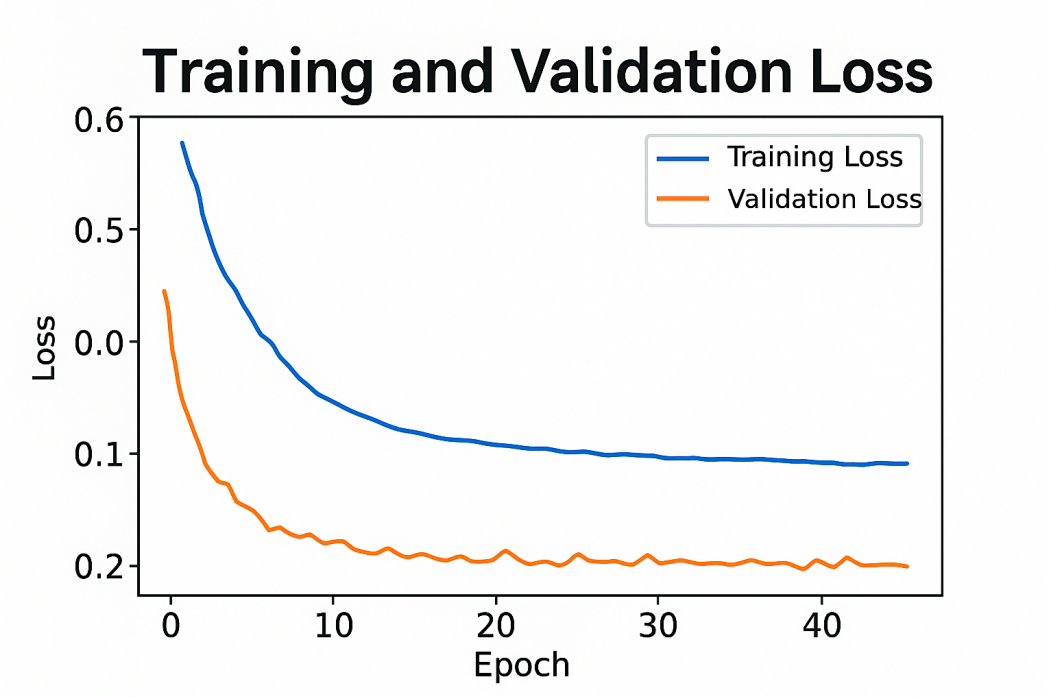
4/5 - Entrenamiento en Keras

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
history = model.fit(train_ds,
                    epochs=20,
                    validation_data=val_ds)
```

- 'compile' asocia pérdida y optimizador.
- 'fit' maneja forward, back-prop y logging.

5/5 - Buenas prácticas

- **Normalizar** datos de entrada.
- **Early-stopping** para evitar sobreajuste.
- Dropout y regularización L2.
- Learning rate schedule / warm-up.



Convolutional Neural Networks (CNN)

- Filtrado local y **parámetros compartidos**.
- Stride, padding & pooling para controlar dimensionalidad.
- Eficientes y con inductive bias para datos con estructura 2D/3D.

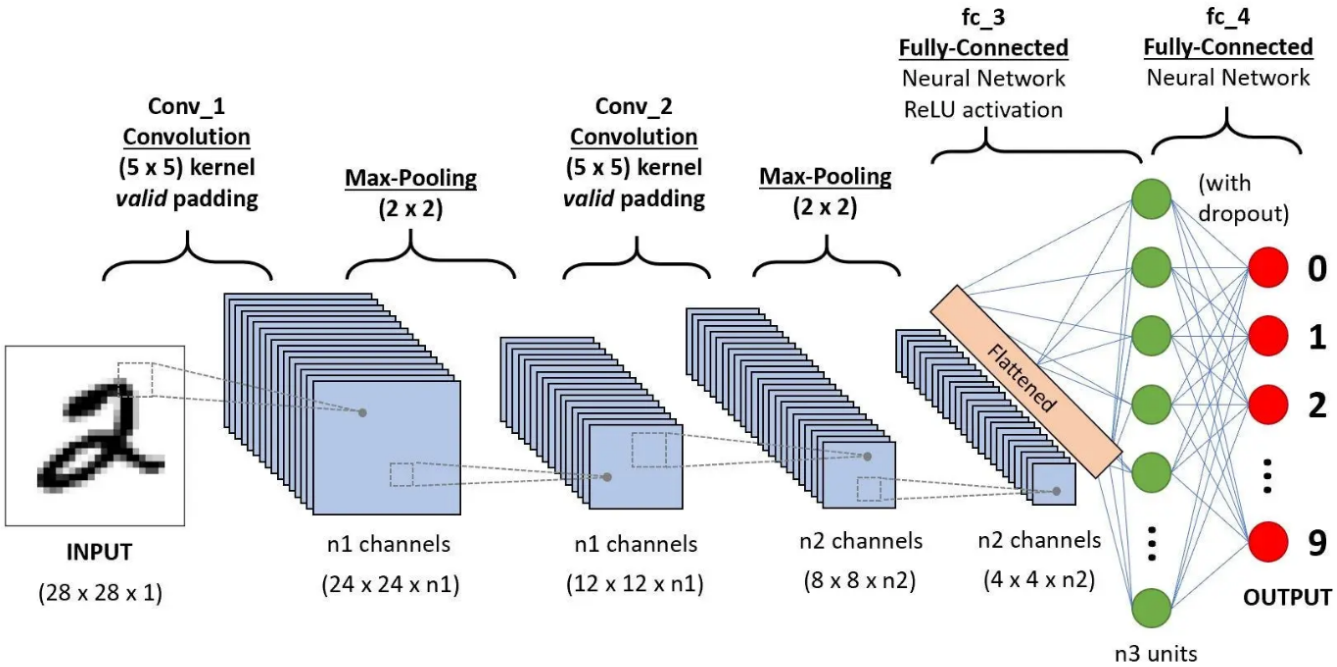


Figura: Proceso standard en CNN

Pooling y arquitectura típica

- Max / Average pooling: reduce resolución, gana invariancia.
- Patrón frecuente: [Conv $\rightarrow$ ReLU]$^{\times n}$ $\rightarrow$ Pool.

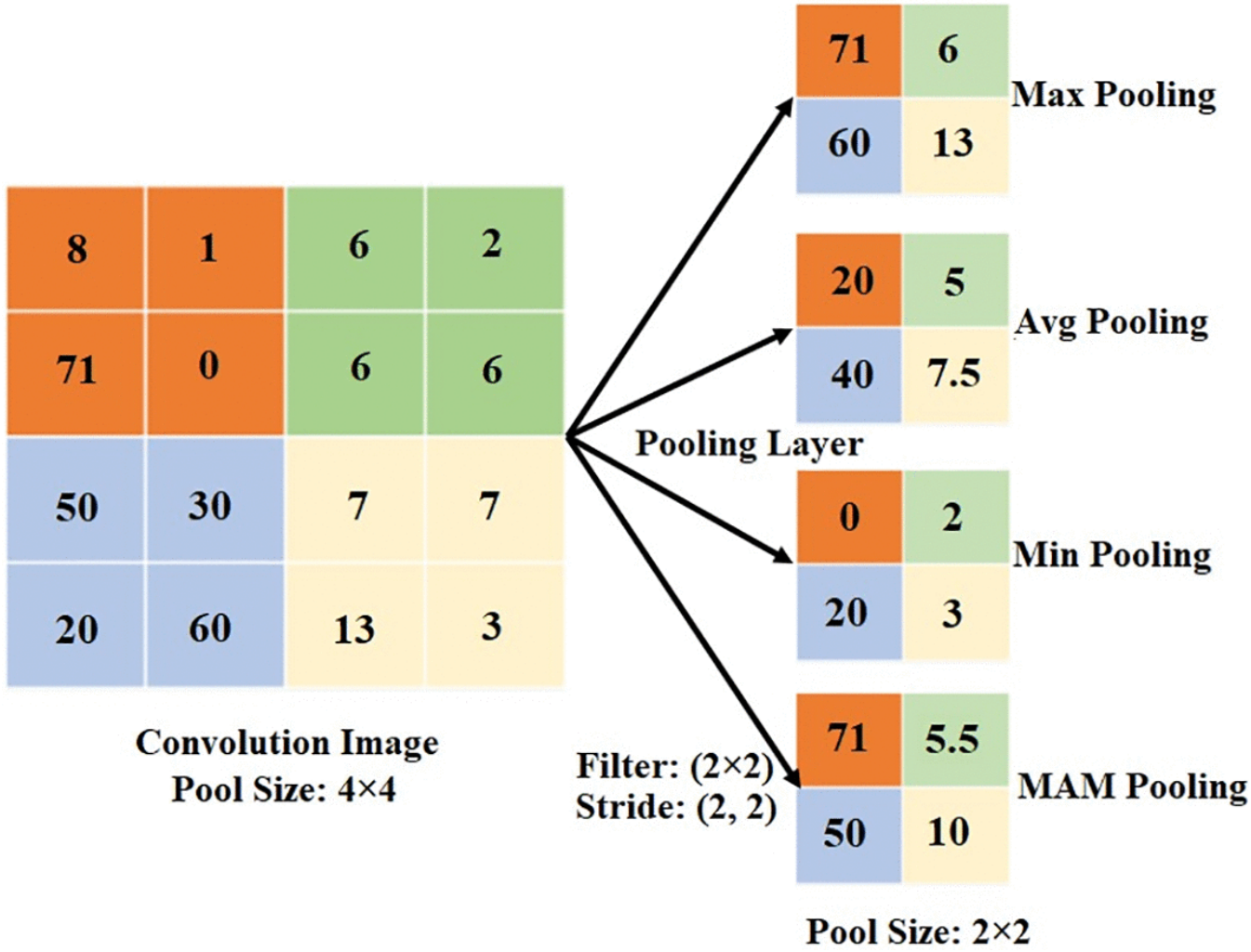


Figura: Cadena Conv–Pool antes de Fully Connected.

Estrategias de regularización

- Dropout
- Weight decay (L_2)
- Data augmentation
- Batch Normalization
- Early Stopping
- Transfer Learning

MLP mínimo en Keras

```python
from keras.models import Sequential
from keras.layers import Dense
import numpy as np

x_train = np.random.random((6000, 10))
y_train = np.random.randint(2, size=(6000, 1))

model = Sequential([
    Dense(32, activation='relu', input_shape=(10,)),
    Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=10, batch_size=64)
```

CNN para MNIST en 15 líneas

```
from keras.datasets import mnist
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D, MaxPool2D, Flatten, Dense

(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = X_train.reshape(-1,28,28,1)/255.
X_test  = X_test.reshape(-1,28,28,1)/255.
y_train = to_categorical(y_train,10)
y_test  = to_categorical(y_test,10)

model = Sequential([
    Conv2D(32,(3,3),activation='relu',input_shape=(28,28,1)),
    MaxPool2D(),
    Flatten(),
    Dense(128,activation='relu'),
    Dense(10,activation='softmax')])
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(X_train,y_train,epochs=5,batch_size=128,
          validation_split=0.1)
```

Conclusiones clave

1. DL aprende representaciones jerárquicas **directamente de los datos**.
2. Las CNN dominan visión, pero hay nuevas arquitecturas (ResNet, Inception, CapsNet).
3. **Transfer Learning** permite entrenar con pocos datos y bajo costo.
4. Regularización y buenas prácticas son esenciales para generalizar.

Para seguir aprendiendo

- Libro gratuito: *Deep Learning*, Goodfellow & Bengio.
- Curso CS231n Stanford (CNNs for Visual Recognition).
- Dataset hub: https://paperswithcode.com/datasets

¿Preguntas?