Deep Learning

Data Science (CC3084) - 2026

Deep Learning

Semestre 02, 2026

Introducción a Deep Learning

El deep learning (aprendizaje profundo) es una rama del aprendizaje automático (machine learning) basada en redes neuronales artificiales con muchas capas. En los últimos años se ha convertido en la técnica dominante para tareas de visión por computadora, procesamiento de lenguaje natural, audio y, cada vez más, para el análisis de series de tiempo.

De la IA al deep learning

Es útil ubicar el deep learning dentro de un conjunto de disciplinas anidadas:

- Inteligencia Artificial (IA): el campo amplio de construir máquinas que exhiben comportamiento inteligente.
- Machine Learning (ML): sistemas que aprenden patrones a partir de datos en lugar de ser programados con reglas explícitas.
- Deep Learning (DL): un subconjunto del ML que usa redes neuronales profundas para aprender representaciones jerárquicas de los datos.

En otras palabras: todo deep learning es machine learning, y todo machine learning es inteligencia artificial, pero no al revés.

Qué cambió: por qué ahora

Las redes neuronales existen desde los años 50 y 80, pero el deep learning despegó alrededor de 2012. Tres factores lo hicieron posible:

- Datos: la digitalización masiva (imágenes, texto, sensores) produjo los grandes volúmenes que estos modelos necesitan.
- Cómputo: las GPU (tarjetas gráficas) permiten entrenar redes con millones de parámetros en tiempos razonables.
- Algoritmos: mejoras como la activación ReLU, mejores inicializaciones, dropout y optimizadores como Adam estabilizaron el entrenamiento de redes profundas.

El cambio de paradigma: aprender las características

La diferencia central con el machine learning clásico es cómo se obtienen las características (features).

- En ML clásico, un experto diseña las características a mano (feature engineering) y luego un modelo simple aprende sobre ellas.
- En deep learning, la red aprende las características automáticamente, en varios niveles de abstracción, directamente desde los datos crudos.

Por ejemplo, en una imagen de un rostro: las primeras capas detectan bordes, las intermedias combinan bordes en ojos y narices, y las últimas reconocen rostros completos. Nadie programó esos detectores: la red los descubrió.

Qué se conoce como Deep Learning

La neurona artificial

La unidad básica es la neurona (o perceptrón). Recibe varias entradas, cada una con un peso, las combina y produce una salida.

El cálculo tiene dos pasos:

$$ z = \sum_{i} w_i x_i + b $$

$$ a = \phi(z) $$

- x_i son las entradas y w_i los pesos que aprende la red.
- b es el sesgo (bias), que desplaza el resultado.
- ϕ es la función de activación, que introduce no linealidad.

Los pesos y sesgos son los parámetros que el modelo ajusta durante el entrenamiento.

Capas y redes

Una red neuronal organiza las neuronas en capas:

- Capa de entrada: recibe los datos (una neurona por característica).
- Capas ocultas: transforman la información; aquí está la "profundidad" del deep learning.
- Capa de salida: produce el resultado (una clase, un número, una probabilidad).

Se llama profunda a una red con varias capas ocultas. Cada capa aprende una representación más abstracta que la anterior.

Funciones de activación

Sin una función de activación no lineal, apilar capas sería inútil: la red completa se reduciría a una simple combinación lineal. Las más comunes:

- ReLU: $\phi(z) = \max(0, z)$. La más usada en capas ocultas; simple y eficiente.
- Sigmoide: $\phi(z) = \frac{1}{1 + e^{-z}}$. Comprime a $(0, 1)$; útil para probabilidades binarias.
- Tanh: comprime a $(-1, 1)$; centrada en cero.
- Softmax: convierte un vector en una distribución de probabilidad; se usa en la salida para clasificación multiclase.

El paso hacia adelante (forward pass)

Predecir consiste en pasar los datos por la red, capa por capa: la salida de una capa es la entrada de la siguiente, hasta llegar a la capa final. A esto se le llama forward pass o propagación hacia adelante.

Cómo aprende una red

El entrenamiento busca los pesos que hacen que las predicciones se parezcan a los valores reales. Tiene tres piezas:

1. Función de pérdida (loss): mide qué tan lejos está la predicción $\hat{y}$ del valor real y. Por ejemplo, el error cuadrático medio para regresión:

$$ L = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 $$

2. Descenso de gradiente: ajusta cada peso en la dirección que reduce la pérdida, controlado por la tasa de aprendizaje η:

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

3. Backpropagation (retropropagación): el algoritmo que calcula eficientemente esos gradientes propagando el error desde la salida hacia atrás, usando la regla de la cadena.

Este ciclo (forward → calcular pérdida → backpropagation → actualizar pesos) se repite muchas veces. Una pasada completa por todos los datos de entrenamiento se llama época (epoch).

Un perceptrón multicapa en código

Con librerías como Keras, definir una red densa es directo:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([
    Dense(64, activation='relu', input_shape=(n_features,)),
    Dense(32, activation='relu'),
    Dense(1)  # salida para regresion
])

model.compile(optimizer='adam', loss='mse')
model.fit(X_train, y_train, epochs=50, batch_size=32)
```

Arquitecturas principales

No todas las redes son iguales; su estructura se adapta al tipo de dato:

- MLP (perceptrón multicapa): capas densas totalmente conectadas. Datos tabulares.
- CNN (redes convolucionales): detectan patrones locales. Imágenes y también series.

- RNN (redes recurrentes): procesan secuencias manteniendo un estado. Texto, audio, series de tiempo.
- LSTM y GRU: variantes de RNN que recuerdan dependencias de largo plazo.
- Transformers: basados en el mecanismo de atención; el estado del arte en lenguaje y, crecientemente, en series.

Discusión sobre Deep Learning

El deep learning es poderoso, pero no es una solución universal. Conviene entender cuándo brilla y cuándo no.

Fortalezas

- Aprende características automáticamente, evitando el costoso feature engineering manual.
- Escala muy bien: con más datos y más cómputo, suele seguir mejorando.
- Alcanza el estado del arte en visión, lenguaje, audio y generación de contenido.
- Es flexible: la misma familia de técnicas se adapta a imágenes, texto, secuencias y más.

Debilidades

- Necesita muchos datos: con conjuntos pequeños, modelos más simples suelen ganar.
- Es costoso: entrenar requiere GPU, tiempo y energía.
- Es una caja negra: cuesta explicar por qué tomó una decisión (baja interpretabilidad).

- Tiene muchos hiperparámetros (capas, neuronas, tasa de aprendizaje) difíciles de ajustar.
- Puede sobreajustar (overfitting) si no se regulariza (dropout, early stopping, más datos).

Deep learning frente a machine learning clásico

No siempre conviene usar deep learning. Una guía práctica:

- Datos pequeños o tabulares con buenas características: modelos clásicos (regresión, árboles, gradient boosting) suelen ser mejores, más rápidos e interpretables.
- Datos grandes y no estructurados (imágenes, texto, audio, señales): el deep learning tiende a dominar.
- Si necesitas explicar cada decisión (medicina, crédito, legal): la interpretabilidad puede pesar más que la exactitud.

Consideraciones éticas y prácticas

- Sesgos: la red aprende los sesgos presentes en los datos; puede amplificar desigualdades.
- Costo energético: entrenar modelos grandes consume mucha energía y tiene huella de carbono.
- Datos y privacidad: requiere grandes cantidades de datos, a veces sensibles.
- Reproducibilidad: los resultados dependen de semillas aleatorias, versiones y hardware.

Deep Learning para Series de Tiempo

Una serie de tiempo es una secuencia de observaciones ordenadas en el tiempo. Su reto particular es que el orden importa y las observaciones cercanas están correlacionadas. El deep learning ofrece modelos diseñados justamente para capturar esas dependencias temporales.

El reto temporal

A diferencia de un dataset tabular, en una serie no podemos barajar las filas: cada valor depende de los anteriores. Un buen modelo debe recordar el pasado para predecir el futuro, y respetar el orden cronológico al entrenar y evaluar.

De serie a problema supervisado: ventanas deslizantes

Para que una red aprenda de una serie, la convertimos en un problema supervisado con la técnica de ventanas deslizantes (sliding window): usamos los últimos k valores para predecir el siguiente.

```
import numpy as np

def crear_ventanas(serie, k):
    X, y = [], []
    for i in range(len(serie) - k):
        X.append(serie[i:i+k])   # ventana de k valores pasados
        y.append(serie[i+k])     # el valor siguiente
    return np.array(X), np.array(y)
```

Con una ventana $k = 3$, la serie `[10, 12, 13, 15, 16]` produce:

- $X = [10, 12, 13] \rightarrow y = 15$
- $X = [12, 13, 15] \rightarrow y = 16$

RNN: redes recurrentes

Una red recurrente procesa la secuencia paso a paso y mantiene un estado oculto h_t que resume lo visto hasta el instante t:

$$ h_t = \phi(W_x x_t + W_h h_{t-1} + b) $$

El estado h_{t-1} se reinyecta en el siguiente paso: así la red "recuerda" el pasado. Es la arquitectura natural para secuencias.

El problema del gradiente que se desvanece

Las RNN simples tienen dificultad para aprender dependencias de largo plazo: al propagar el error muchos pasos hacia atrás, los gradientes se hacen diminutos (vanishing gradient) y la red olvida lo que pasó hace tiempo. Este problema motivó arquitecturas con memoria.

LSTM y GRU

Las LSTM (Long Short-Term Memory) y las GRU (Gated Recurrent Unit) resuelven el olvido con compuertas (gates): mecanismos que deciden qué información guardar, qué olvidar y qué dejar pasar. Mantienen una memoria de largo plazo, ideal para series con estacionalidad o tendencias prolongadas.

Un modelo LSTM para pronóstico en Keras:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

model = Sequential([
    LSTM(50, activation='tanh', input_shape=(k, 1)),
    Dense(1)
])

model.compile(optimizer='adam', loss='mse')
model.fit(X_train, y_train, epochs=20, batch_size=16)
```

Otras arquitecturas para series

- CNN 1D: aplican convoluciones a lo largo del tiempo para detectar patrones locales (picos, formas). Rápidas y efectivas.
- Transformers: usan atención para relacionar cualquier par de instantes de la secuencia, capturando dependencias de muy largo plazo. Son la frontera actual en pronóstico.
- Modelos híbridos: combinan CNN + LSTM, o incorporan variables externas (clima, feriados) como entradas adicionales.

Consideraciones prácticas

- Escalado: normaliza la serie (por ejemplo a $[0, 1]$) antes de entrenar; las redes son sensibles a la escala.
- División temporal: nunca uses datos del futuro para entrenar. El conjunto de prueba debe ser el tramo final de la serie, no una muestra aleatoria.
- Horizonte de pronóstico: predecir el siguiente valor (one-step) es más fácil que predecir varios pasos (multi-step), donde el error se acumula.
- Métricas: se evalúa con MAE (error absoluto medio), RMSE (raíz del error cuadrático medio) o MAPE (error porcentual absoluto medio).

¿Cuándo usar deep learning en series?

- Métodos clásicos como ARIMA o suavizado exponencial siguen siendo excelentes para series cortas, univariadas y con estructura clara.
- El deep learning gana cuando hay muchas series o datos, relaciones no lineales, múltiples variables, o patrones complejos que los modelos clásicos no capturan.

La recomendación práctica: empieza con un modelo simple como línea base, y sube a deep learning solo si los datos y el problema lo justifican.