Análisis Comparativo de Arquitecturas de Deep Learning

Infografía técnica

Arquitecturas Densas, Convolucionales, Recurrentes y Basadas en Atención

1. Notación general y núcleo común

Este póster resume un **análisis exhaustivo** de las principales arquitecturas de aprendizaje profundo (Densas, Convolucionales, Recurrentes y basadas en Atención), enfatizando que todas comparten un **modelo matemático fundamental** y un mismo esquema de entrenamiento basado en **backpropagation** y **descenso de gradiente**.

Símbolos básicos usados en todas las arquitecturas

Símbolo	Tipo	Significado / Explicación
l	Entero	Índice de la capa actual. Recorre de 1 (primera capa oculta) a L (capa de salida).
$\mathbf{x}$	Vector	Vector de entrada: datos iniciales que se pasan a la red.
$W^{(l)}$	Matriz	Matriz de pesos de la capa l: parámetros que la red aprende. Mide la importancia de las entradas. $W^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$.
$\mathbf{b}^{(l)}$	Vector	Vector de sesgos (*bias*) de la capa l: desplaza la activación y añade flexibilidad. $\mathbf{b}^{(l)} \in \mathbb{R}^{n_l}$.
$\mathbf{z}^{(l)}$	Vector/Tensor	**Entrada neta** (pre-activación) de la capa l: $\mathbf{z}^{(l)} = W^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$.
$\mathbf{a}^{(l)}$	Vector/Tensor	**Salida de la capa** l (activación): $\mathbf{a}^{(l)} = f\big(\mathbf{z}^{(l)}\big)$.
$f(\cdot)$	Función	Función de activación (ReLU, Sigmoid, Tanh, Softmax...). Introduce la no-linealidad.
$L(\cdot)$	Escalar	Función de pérdida (*loss*): mide el error entre la predicción $\hat{y}$ y la verdad y.
$\hat{y}$, y	Escalar / vector	$\hat{y}$: predicción de la red; y: etiqueta verdadera.
η	Escalar	Tasa de aprendizaje: tamaño del paso en la dirección del gradiente.

2. Backpropagation: desglose profundo

El objetivo del entrenamiento es calcular los gradientes $\dfrac{\partial L}{\partial W^{(l)}}$ ("cuánto debe cambiar un peso específico para reducir la pérdida") y actualizar los parámetros.

1. Propagación hacia adelante (Forward Pass)

$$\mathbf{z}^{(l)} = W^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}, \qquad \mathbf{a}^{(l)} = f\big(\mathbf{z}^{(l)}\big).$$

La salida de la red se denota $\hat{y} = \mathbf{a}^{(L)}$ y se calcula la pérdida $L(\hat{y}, y)$.

2. Error en la capa de salida

$$\delta^{(L)} = \frac{\partial L}{\partial \mathbf{z}^{(L)}} = \frac{\partial L}{\partial \mathbf{a}^{(L)}} \odot f'\big(\mathbf{z}^{(L)}\big),$$

donde $\odot$ es el producto elemento a elemento (Hadamard).

3. Retropropagación hacia capas ocultas

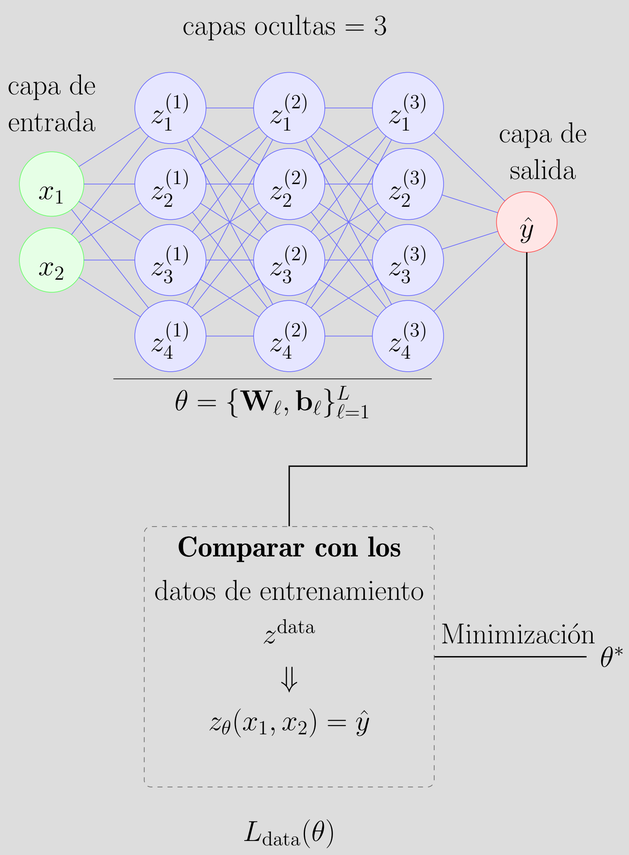
$$\delta^{(l)} = \big(W^{(l+1)}\big)^T \delta^{(l+1)} \odot f'\big(\mathbf{z}^{(l)}\big), \qquad l = L-1, \ldots, 1.$$

4. Gradientes de los pesos y actualización

$$\frac{\partial L}{\partial W^{(l)}} = \delta^{(l)}\big(\mathbf{a}^{(l-1)}\big)^T, \qquad W^{(l)} \leftarrow W^{(l)} - \eta \frac{\partial L}{\partial W^{(l)}}.$$

Este **esquema de backpropagation** es el núcleo común de MLP, CNN, RNN/LSTM y Transformers.

Esquema de una red neuronal clásica (MLP)



Esquema de red densa (MLP): entradas (x_1, x_2), capas ocultas y salida $\hat{y} = z_\theta(x_1, x_2)$.

3. Funciones de activación: el elemento no lineal

Las funciones de activación $f(\cdot)$ se aplican a la entrada neta z de cada neurona para decidir si se "activa" y con qué intensidad. Sin ellas, la red sería puramente lineal y no podría aprender fronteras complejas.

Función	Fórmula	Propósito / Ventaja	Problema principal
Sigmoid (Logística)	$f(z) = \dfrac{1}{1+e^{-z}}$	Comprime la salida a $(0, 1)$, lo que permite interpretarla como **probabilidad**. Muy usada en la **capa de salida** de clasificación binaria.	**Gradiente desvanecedor**: para $\lvert z\rvert$ grande, $f'(z)$ se aproxima a 0, lo que detiene el aprendizaje en capas profundas.
Tanh (Tangente hiperbólica)	$f(z) = \tanh(z)$	Salida en $(-1, 1)$ y centrada en cero, lo que ayuda a la simetría de los gradientes y suele funcionar mejor que Sigmoid en capas ocultas.	También sufre **gradiente desvanecedor**: la saturación en los extremos reduce la magnitud de $f'(z)$.
ReLU (Rectified Linear Unit)	$f(z) = \max(0, z)$	Estándar en muchas capas ocultas. Es computacionalmente muy eficiente y reduce el gradiente desvanecedor para $z > 0$, permitiendo redes más profundas.	**Dying ReLU**: si la entrada es siempre negativa, el gradiente es 0 y la neurona "muere" (deja de aprender).
Softmax	$f(z)_i = \dfrac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$	Convierte las puntuaciones (*logits*) en una **distribución de probabilidad** que suma 1. Esencial en la salida de **clasificación multiclase**.	Se utiliza solo en la capa de salida. Su derivada completa es una matriz (Jacobiano), aunque se simplifica mucho al combinarla con la pérdida **Cross-Entropy**.

4. Arquitectura densa (Fully Connected / MLP)

¿Cómo funciona por dentro? La capa densa implementa una transformación lineal afín seguida de una activación no lineal. Cada neurona de la capa l recibe entradas de *todas* las neuronas de la capa $l-1$.

Modelo matemático fundamental

$$\mathbf{z}^{(l)} = W^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}, \qquad \mathbf{a}^{(l)} = f\big(\mathbf{z}^{(l)}\big).$$

Dimensiones típicas:

$$W^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}, \quad \mathbf{a}^{(l-1)} \in \mathbb{R}^{n_{l-1}}.$$

¿Por qué se usa?

- Es el **bloque constructor universal** del Deep Learning (Teorema de Aproximación Universal).
- Se utiliza en la **toma de decisiones final** (clasificación/regresión).
- Sirve como bloque genérico para transformar representaciones internas.

Características claves y limitaciones

- No hay parámetros compartidos: cada conexión tiene un peso distinto.
- No explota estructura espacial o temporal: trata cada componente (píxel, palabra, etc.) como una característica independiente.

5. Arquitectura Convolucional (CNN)

¿Cómo funciona por dentro? En lugar de una multiplicación matricial completa, se utiliza la **convolución**. Un filtro (kernel) pequeño se desliza sobre la entrada y hace un *producto punto* local en cada posición.

¿Por qué se hace?

1. **Compartición de pesos**: el mismo filtro W se aplica en toda la entrada $\Rightarrow$ muchos menos parámetros.
2. **Conexiones locales**: el filtro solo "ve" una región local, capturando bordes, texturas y patrones locales.
3. **Invarianza traslacional**: si un patrón (p. ej. un gato) se desplaza un poco en la imagen, el mismo filtro puede seguir detectándolo.

Modelo matemático de la convolución 2D
Para un solo filtro en la capa l:

$$z_{i,j,k}^{(l)} = b_k^{(l)} + \sum_{c=1}^{C_{l-1}} \sum_{m=1}^{F} \sum_{n=1}^{F} W_{m,n,c,k}^{(l)}\, a_{i+m,\, j+n,\, c}^{(l-1)},$$

6. Arquitectura Recurrente (RNN / LSTM)

¿Cómo funciona por dentro? La clave es el **estado oculto** $\mathbf{h}_t$. En una RNN simple, la salida en el tiempo t se retroalimenta como entrada en el tiempo $t+1$, creando una memoria de la secuencia:

$$\mathbf{h}_t = \phi\big(W_h\mathbf{h}_{t-1} + W_x\mathbf{x}_t + \mathbf{b}\big).$$

¿Por qué se usa?

- Esencial para datos **secuenciales** (texto, audio, series de tiempo).
- El orden y el contexto temporal son críticos.

Problema de la RNN simple Sufre el **gradiente desvanecedor** (y explosivo), lo que dificulta aprender dependencias a largo plazo. Por eso se emplean variantes como **LSTM** y **GRU**.

Modelo matemático fundamental de la célula LSTM
La LSTM introduce un **estado de célula** $\mathbf{C}_t$ (memoria de largo plazo) controlado por compuertas logísticas (valores entre 0 y 1):

$$\begin{aligned}
\text{Compuerta de olvido:} \quad & \mathbf{f}_t = \sigma\big(W_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f\big), \\
\text{Compuerta de entrada:} \quad & \mathbf{i}_t = \sigma\big(W_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i\big), \\
& \tilde{\mathbf{C}}_t = \tanh\big(W_C[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_C\big), \\
\text{Actualización del estado de célula:} \quad & \mathbf{C}_t = \mathbf{f}_t \odot \mathbf{C}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{C}}_t, \\
\text{Compuerta de salida:} \quad & \mathbf{o}_t = \sigma\big(W_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o\big), \\
\text{Estado oculto:} \quad & \mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{C}_t).
\end{aligned}$$

Interpretación:

- $\mathbf{f}_t$ decide qué olvidar del pasado ($\mathbf{C}_{t-1}$).
- $\mathbf{i}_t$ y $\tilde{\mathbf{C}}_t$ deciden qué nueva información guardar.
- $\mathbf{o}_t$ controla qué parte de la memoria se expone como estado oculto $\mathbf{h}_t$.

7. Arquitectura basada en Atención (Transformers)

¿Cómo funciona por dentro? El Transformer elimina la recurrencia y utiliza **Auto-Atención (Self-Attention)** para modelar dependencias largas en paralelo.

Construcción de consultas, claves y valores (Q, K, V)
Dada una secuencia de entrada $X \in \mathbb{R}^{S \times d}$ (longitud S, dimensión d):

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V,$$

donde W^Q, W^K, W^V son matrices de pesos aprendidas y $Q, K, V \in \mathbb{R}^{S \times d_k}$.

Atención de producto punto escalado

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V.$$

- QK^T mide la similitud entre cada *query* y cada *key*.
- El factor $1/\sqrt{d_k}$ estabiliza las magnitudes antes del `softmax`.
- El `softmax` produce pesos de atención (probabilidades).
- La multiplicación por V da una combinación ponderada de los valores.

Atención multi-cabeza (Multi-Head Attention)
En la práctica se usan varias cabezas en paralelo:

$$\text{head}_i = \text{Attention}\big(QW_i^Q,\ KW_i^K,\ VW_i^V\big),$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\, W^O.$$

Cada cabeza aprende un tipo distinto de relación (sintáctica, semántica, posicional, etc.).

8. Resumen comparativo de arquitecturas

Arquitectura	Aplicaciones clave	Rol principal
Densa (MLP)	Datos tabulares; capas finales en redes profundas; salida de modelos de clasificación y regresión.	Bloque universal genérico. Transforma vectores de características en decisiones finales (clases, valores continuos).
CNN	Visión por computador (clasificación, segmentación, detección); audio (espectrogramas); datos 2D/3D estructurados.	Extrae jerárquicamente características espaciales y patrones locales. Aprovecha la estructura de las imágenes y reduce parámetros.
RNN / LSTM	Texto, voz, traducción automática, modelado de lenguaje, series de tiempo y secuencias en general.	Modela dependencias temporales y contexto histórico mediante estados ocultos y memoria (célula LSTM). Captura orden y dinámica temporal.
Transformers	PLN, visión, audio, grandes modelos de lenguaje (LLM), modelos multimodales.	Captura dependencias de largo alcance vía auto-atención, eliminando recurrencia y permitiendo entrenamiento altamente paralelo en GPU/TPU.