Fundamentos del Deep Learning

INTELIGENCIA ARTIFICIAL

Autor: Grupo 2

- Gonzales Marca Jesus Miguel
- Suarez Bautista Pablo Israel
- Pachas Oshiro Kojiro Andre
- Lliuya Villagaray Joaquin Lazaro

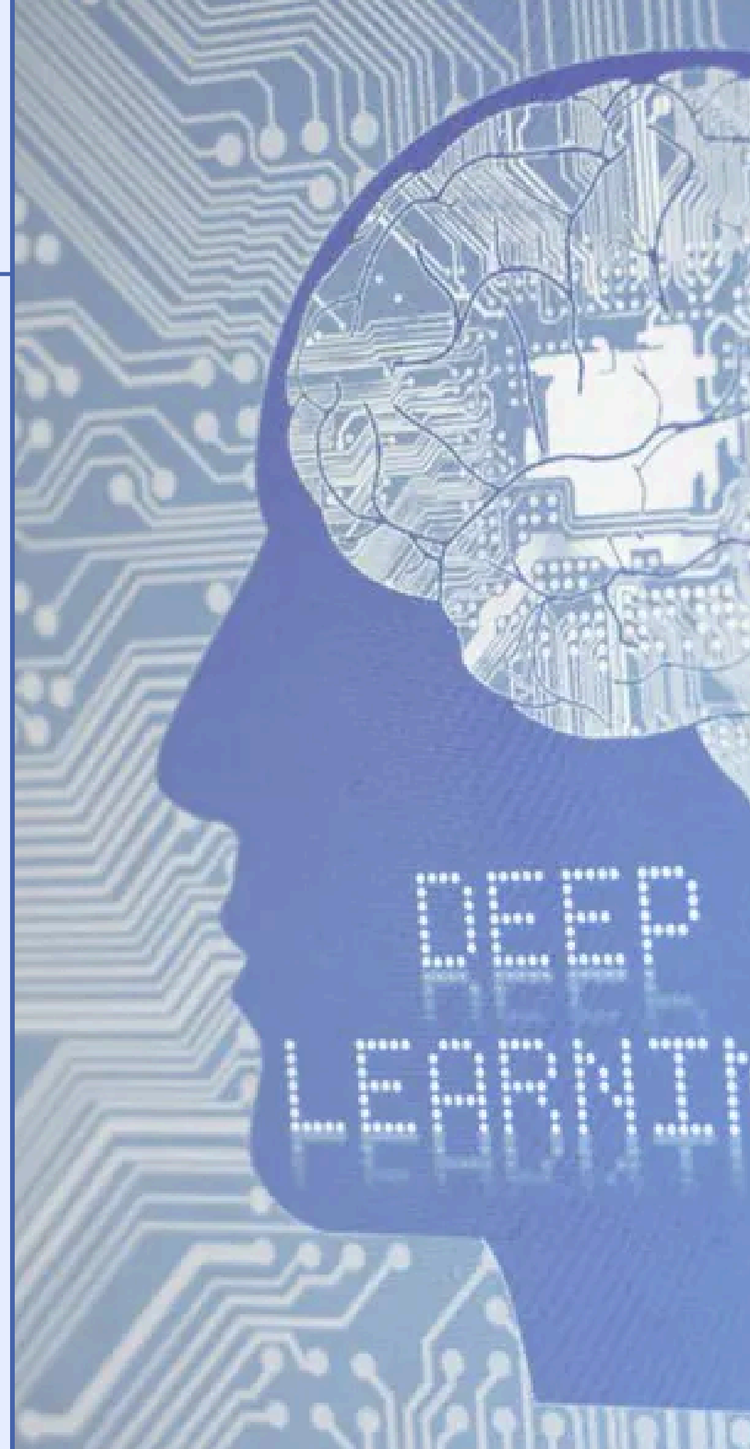
Introducción

El Deep Learning, o aprendizaje profundo, es una subdisciplina dentro del campo del Machine Learning que se caracteriza por el uso de redes neuronales artificiales con múltiples capas ocultas, denominadas "profundas", que permiten modelar relaciones complejas entre datos. A diferencia de otros enfoques tradicionales, el Deep Learning aprende directamente a partir de grandes volúmenes de datos, sin que sea necesario definir manualmente las reglas o características que debe aprender. Este enfoque ha sido fundamental para el desarrollo de tecnologías como el reconocimiento facial, los asistentes virtuales, los traductores automáticos y los sistemas de recomendación.

Evolución histórica

Década de 1940-1950: Origen de las neuronas artificiales y el perceptrón

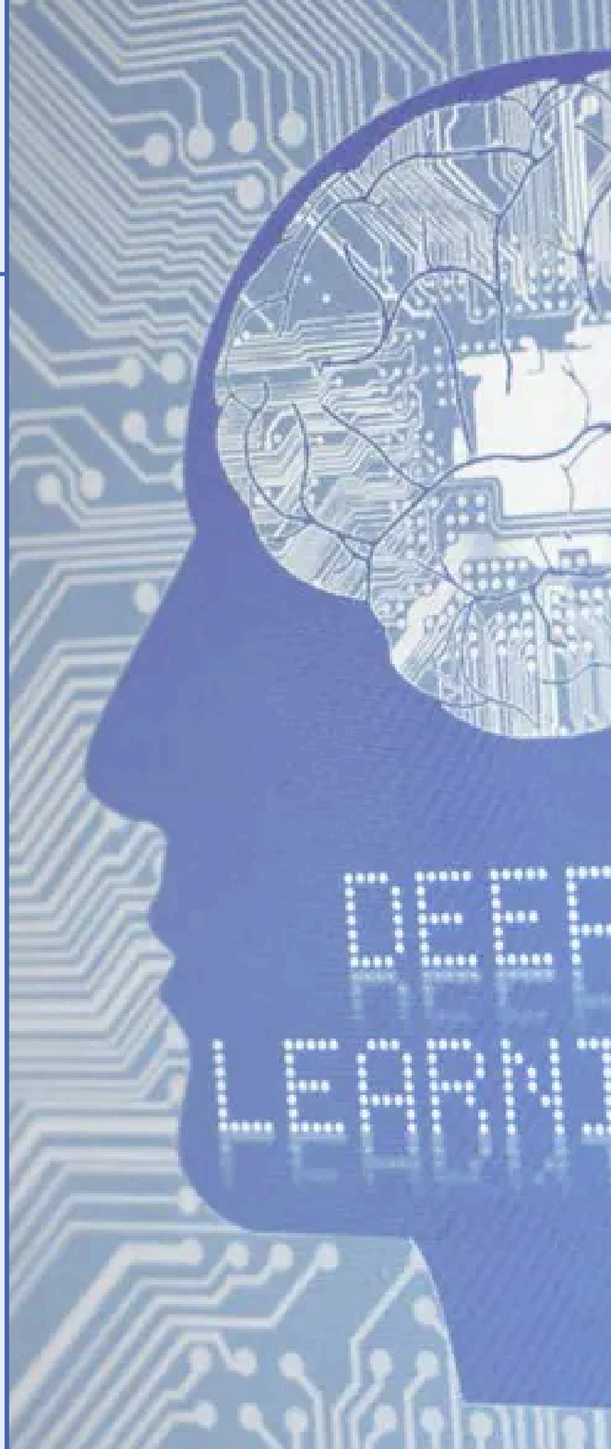
- Modelo de McCulloch-Pitts (1943): Warren McCulloch y Walter Pitts propusieron uno de los primeros modelos formales de una neurona artificial. Su modelo era una simplificación matemática de cómo una neurona biológica procesa señales, usando funciones lógicas binarias (activación o no activación). Fue fundamental para establecer que sistemas formales podían imitar procesos neuronales.
- Perceptrón (1958): Frank Rosenblatt desarrolló el perceptrón, la primera red neuronal artificial capaz de aprender mediante un algoritmo simple de ajuste de pesos. El perceptrón era una red de una sola capa, lo que limitaba su capacidad para resolver problemas no lineales. Aun así, fue un hito porque introdujo la idea de aprendizaje automático mediante ejemplos.



Evolución histórica

Década de 1980: La revolución del aprendizaje profundo con retropropagación

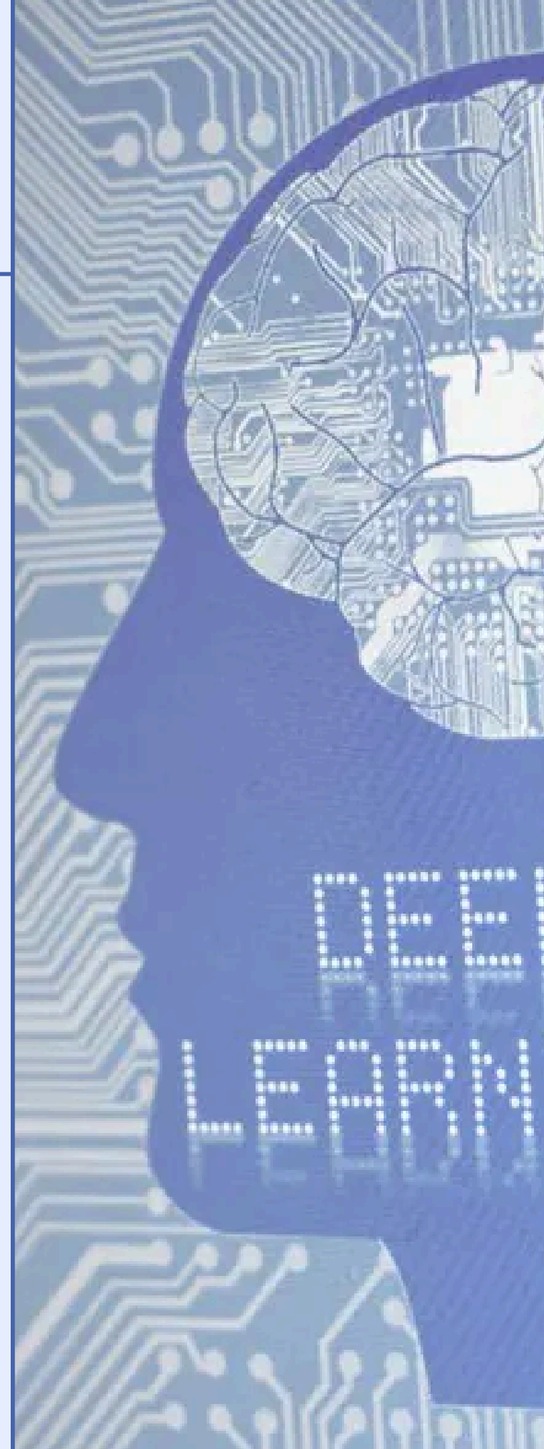
- **Retropropagación** (1986): Aunque el algoritmo de retropropagación ya existía en teoría, fue en esta década cuando se popularizó gracias a los trabajos de Rumelhart, Hinton y Williams. Este algoritmo permitió entrenar redes neuronales multicapa (conocidas como perceptrones multicapa o MLP), superando las limitaciones del perceptrón simple.
- **Importancia:** La retropropagación es un método para calcular el gradiente del error de salida con respecto a los pesos internos de la red, y así ajustar esos pesos eficientemente para reducir el error. Esto permitió que las redes neuronales aprendieran representaciones más complejas y jerárquicas, lo que es la base del deep learning.
- **Limitaciones de la época:** A pesar de los avances, las redes eran aún relativamente pequeñas por limitaciones computacionales, y los datos disponibles eran limitados, lo que frenó un poco la adopción masiva.



Evolución histórica

Década 2000-2010: Resurgimiento gracias a computación y big data

- **Poder computacional (GPUs):** La llegada y adopción masiva de GPUs (unidades de procesamiento gráfico) permitió realizar cálculos paralelos mucho más rápido, acelerando el entrenamiento de redes neuronales profundas.
- **Big Data:** La disponibilidad de grandes conjuntos de datos (por ejemplo, imágenes etiquetadas, texto, video) proporcionó el combustible necesario para entrenar modelos con muchas capas y millones de parámetros.
- **Avances clave:** Aparecieron arquitecturas como las redes convolucionales profundas (CNNs), que revolucionaron la visión por computador, y las redes recurrentes (RNNs), que mejoraron el procesamiento de secuencias y lenguaje.
- **Casos emblemáticos:** En 2012, AlexNet, una red convolucional profunda, ganó el concurso ImageNet con una precisión mucho mayor que los métodos previos, marcando el inicio del "boom" moderno del deep learning.



Evolución histórica

Actualidad: Deep learning como base de tecnologías de punta

- **Modelos de lenguaje natural:** Modelos como GPT (Generative Pretrained Transformer), ChatGPT y otros transformadores usan arquitecturas de deep learning para entender y generar texto de forma avanzada, siendo la base de chatbots, asistentes virtuales y traducción automática.
- **Visión por computador:** Redes convolucionales y arquitecturas derivadas se usan en reconocimiento facial, diagnóstico médico, vehículos autónomos, etc.
- **Generación de contenido:** Modelos como DALL·E generan imágenes a partir de texto, mezclando deep learning y multimodalidad.
- **Sistemas de recomendación:** Plataformas como Netflix, YouTube o Spotify usan deep learning para personalizar contenido y mejorar la experiencia del usuario.
- **Futuro cercano:** Se exploran redes neuronales cada vez más complejas, multimodales (que entienden y generan texto, imagen, audio simultáneamente) y con capacidad para razonar y aprender con menos datos.

Caracteríticas principales

- Arquitecturas profundas: Redes con muchas capas ocultas que permiten representar funciones altamente no lineales.
- Aprendizaje jerárquico: Aprende representaciones desde lo simple (bordes en una imagen) hasta lo complejo (rostros, objetos).
- Extracción automática de características: No requiere ingeniería manual de atributos.
- Gran capacidad de generalización: Cuando se entrena con suficiente información, supera a muchos métodos tradicionales.

¿Cómo aprende una red neuronal?

- Cada neurona recibe entradas, las pondera mediante pesos, las suma y aplica una función de activación.
- La red predice una salida, compara con el valor real (función de pérdida), y ajusta los pesos usando backpropagation y gradient descent.
- Este proceso se repite en múltiples epochs hasta que la red minimiza el error y generaliza bien sobre nuevos datos.

Comparación con el Machine Learning clásico

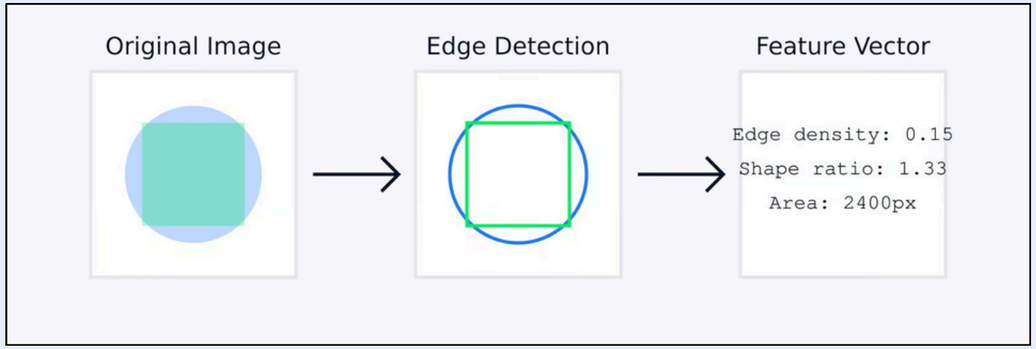
Deep Learning vs Machine Learning

Característica	Aprendizaje Supervisado Clásico	Deep Learning
Extracción de características	Manual (Ingeniería de atributos)	Automática (aprende de datos)
Requiere muchos datos	No necesariamente	Sí, requiere grandes volúmenes
Rendimiento en tareas complejas	Limitado	Muy alto con datos adecuados
Interpretabilidad	Alta (modelos más simples)	Baja
Tiempo de entrenamiento	Rápido (en general)	Lento y computacionalmente costoso

Extracción de Características

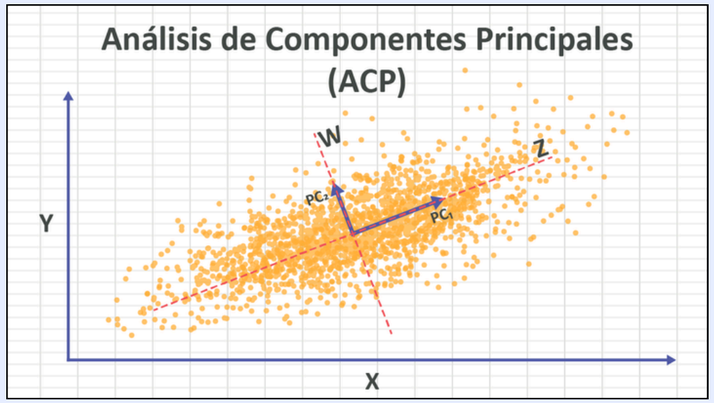
Manual

Ingeniería manual de funciones:
Implica utilizar la experiencia del dominio para identificar y crear rasgos relevantes a partir de datos brutos. Este enfoque práctico se basa en nuestra comprensión del problema y de los datos para elaborar características significativas.



Automática

Extracción automatizada de rasgos:
Utiliza algoritmos para descubrir y crear rasgos sin orientación humana explícita. Estos métodos son especialmente útiles cuando se trata de conjuntos de datos complejos



Otro método
Autocodificadores: Son redes neuronales que aprenden representaciones comprimidas de los datos, captando relaciones no lineales

Requisitos de datos y procesamiento

Los requerimientos para cada modelo de lenguajes varían conforme a la necesidad del problema. De la siguiente manera serían distribuido estos modelos:

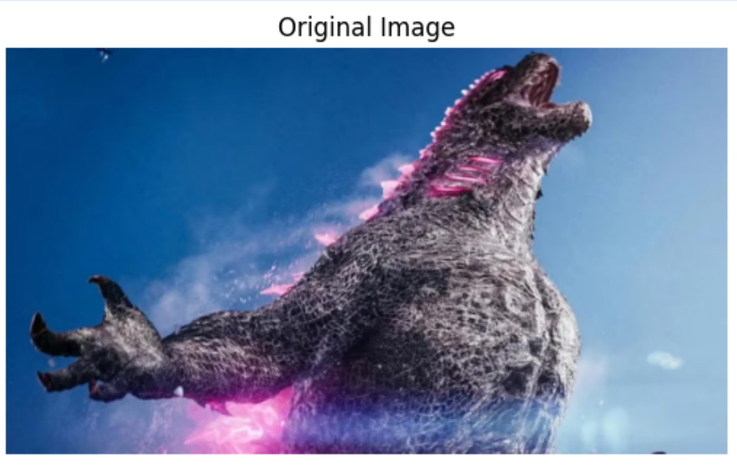
Enfoque	Requisitos de Datos	Requisitos de Procesamiento
Machine Learning	Requiere menos datos (puede funcionar con cientos o pocos miles de muestras). Necesita que las características sean extraídas manualmente.	• Computacionalmente ligero. • Entrenamiento rápido en CPU.
Deep Learning	Requiere gran cantidad de datos para generalizar (decenas de miles o millones). Extrae características automáticamente.	• Alto costo computacional. • Necesita GPU para un entrenamiento eficiente. • Entrenamiento más lento pero escalable.

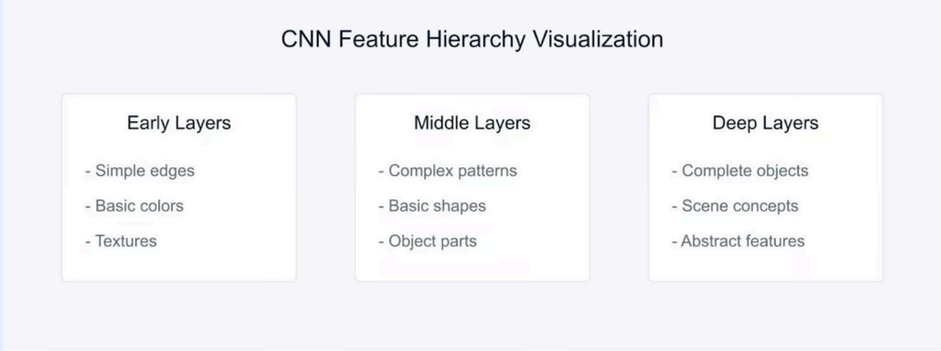
Caso de Ejemplo de extracción de características

Actualmente existen 3 técnicas de extracción

- Extracción de características de la imagen
- Extracción de características de audio
- Extracción de características de series temporales

Para este ejemplo se realizará la extracción de características de la imagen en sus diferentes tipos de extracción.






```
# Load the image
image = cv2.imread('godzilla.jpg')

if image is None:
    print("Error: Could not load the image. Please ensure 'godzilla.jpg' is in the correct directory.")
else:
    # Convert BGR to RGB (OpenCV loads in BGR format)
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # Display the original image
    plt.imshow(image_rgb)
    plt.title('Original Image')
    plt.axis('off')
    plt.show()
```

Caso de Ejemplo de extracción de características

Se aplican 2 filtros

```
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

```
blurred = cv2.GaussianBlur(gray_image, (5, 5), 0)
```

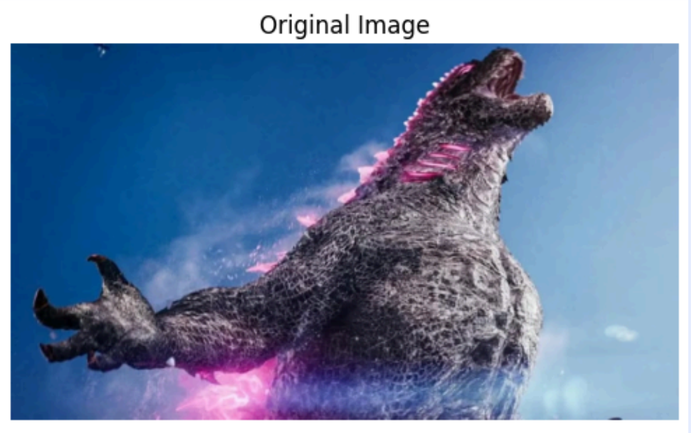
Por último, aplicamos el algoritmo de detección de bordes Canny y visualicemos los resultados:

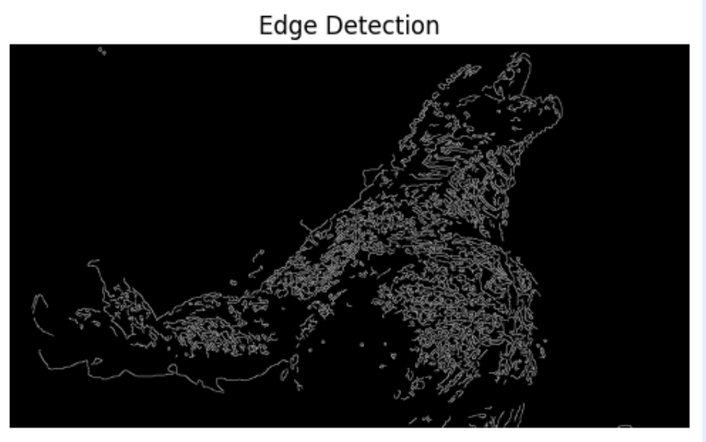
```
edges = cv2.Canny(blurred, threshold1=100, threshold2=200)

# Display the results
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(image_rgb)
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(edges, cmap='gray')
plt.title('Edge Detection')
plt.axis('off')

plt.tight_layout()
plt.show()
```





Caso de Ejemplo de extracción de características

Se importarán las librerías cv2, numpy y matplotlib.pyplot para subir un archivo jpg y para realizar las transformaciones necesarias a la imagen. Además, se implementarán librerías para la ejecución de la IA.

Este código utiliza una red neuronal convolucional (CNN) para analizar y visualizar las activaciones de los filtros en la primera capa convolucional.

```
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
image_resized = cv2.resize(image_rgb, (64, 64))  # Tamaño fijo para la CNN
image_input = image_resized / 255.0              # Normalizar
image_input = np.expand_dims(image_input, axis=0)  # (1, 64, 64, 3)

# 3. Construir red CNN con Functional API
input_img = Input(shape=(64, 64, 3))
x = Conv2D(16, (3, 3), activation='relu')(input_img)
x = MaxPooling2D((2, 2))(x)
x = Conv2D(32, (3, 3), activation='relu')(x)
x = MaxPooling2D((2, 2))(x)
x = Conv2D(64, (3, 3), activation='relu')(x)

model = Model(inputs=input_img, outputs=x)
```

Caso de Ejemplo de extracción de características

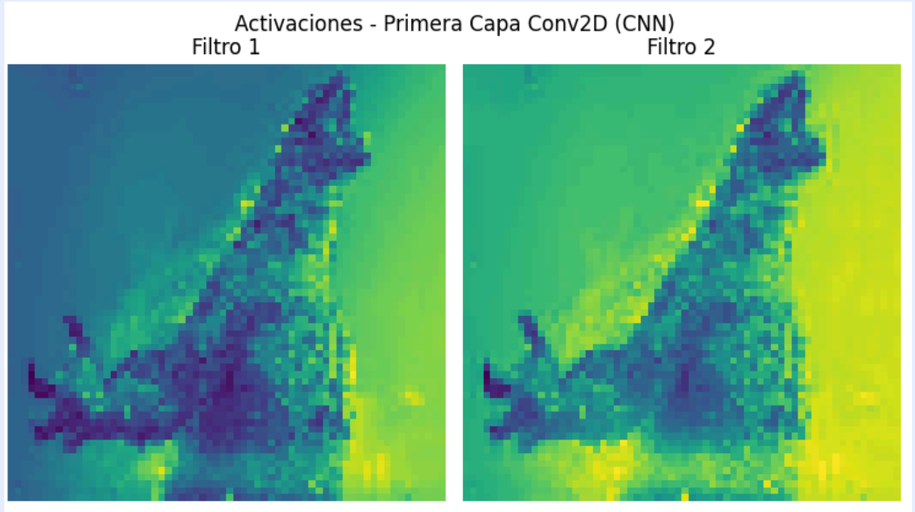
Extrae las salidas de cada capa y muestra los mapas de activación de los primeros 8 filtros de la primera capa convolucional. Cada subplot representa cómo un filtro específico responde a características de la imagen

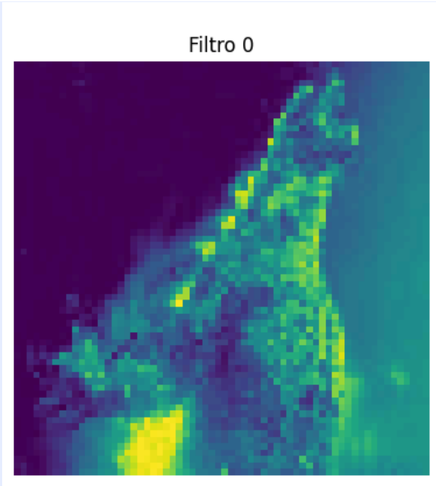
```
# 4. Crear modelo intermedio para ver activaciones
activation_model = Model(inputs=input_img, outputs=[layer.output for layer in model.layers])

# 5. Obtener activaciones
activations = activation_model.predict(image_input)
```

```
# 6. Visualizar activaciones de la primera capa (Conv2D con 16 filtros)
first_layer_activation = activations[0]
num_filters = first_layer_activation.shape[-1]

plt.figure(figsize=(15, 8))
for i in range(min(num_filters, 8)):  # Mostrar primeros 8 filtros
  plt.subplot(2, 4, i + 1)
  plt.imshow(first_layer_activation[0, :, :, i], cmap='viridis')
  plt.title(f'Filtro {i}')
  plt.axis('off')
plt.suptitle('Activaciones - Primera Capa Conv2D (CNN)')
plt.tight_layout()
plt.show()
```

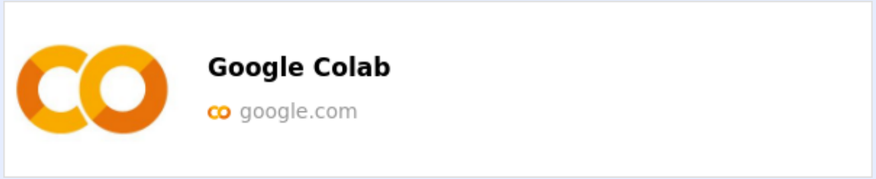




Aplicaciones y Limitaciones del Deep Learning

Aplicaciones destacadas

- Visión por computadora: Las redes neuronales convolucionales (CNNs) permiten identificar objetos, rostros, patrones y escenas en imágenes y videos. Se utilizan en vigilancia, automóviles autónomos, diagnóstico médico por imágenes, reconocimiento facial, y más.
- Procesamiento del lenguaje natural (NLP): Modelos como BERT, GPT y T5 permiten comprender, traducir, resumir y generar texto humano. Se emplean en asistentes virtuales, chatbots, análisis de sentimiento y motores de búsqueda.
- Reconocimiento de voz y síntesis de audio: Modelos como DeepSpeech de Mozilla o Whisper de OpenAI permiten convertir voz en texto con alta precisión. A su vez, redes generativas como WaveNet permiten sintetizar voces humanas casi indistinguibles de las reales.



Aplicaciones y Limitaciones del Deep Learning

Ejemplos reales (industria, salud, transporte, etc.)

- Industria: En la manufactura, se emplean redes neuronales para inspección de calidad mediante imágenes, predicción de mantenimiento de maquinaria (predictive maintenance) y optimización de la cadena de suministro.
- Salud: El deep learning permite diagnosticar enfermedades a partir de radiografías, resonancias o incluso patrones en el habla y escritura. Ejemplo: detección de cáncer de mama con precisión superior a radiólogos (Estudio de Google Health, 2020).
- Transporte: En los vehículos autónomos, los sistemas de conducción utilizan deep learning para interpretar señales de tráfico, peatones, carriles y obstáculos en tiempo real.
- Finanzas: Se emplea para detectar fraudes, analizar riesgo crediticio, predecir movimientos del mercado y automatizar asesoramiento financiero mediante asistentes virtuales.
- Agricultura: Uso de drones con visión por computadora para detectar plagas, enfermedades o deficiencias en cultivos.

Aplicaciones y Limitaciones del Deep Learning

Requiere grandes cantidades de datos

Para que los modelos generalicen bien, necesitan grandes volúmenes de datos etiquetados. Esto no solo es costoso, sino que en algunos sectores (como medicina o derecho) puede ser difícil de obtener por privacidad o escasez.

Alta demanda computacional

El entrenamiento de modelos profundos requiere GPUs o TPUs de alto rendimiento, consumo energético elevado y tiempos prolongados, lo que representa una barrera para muchos desarrolladores e instituciones pequeñas.

Problemas de explicabilidad

Muchos modelos funcionan como cajas negras: generan predicciones precisas, pero no es claro cómo llegaron a esas conclusiones. Esto es un riesgo en sectores como salud, justicia o finanzas, donde la interpretabilidad es clave.

Ética y riesgos potenciales

Privacidad y vigilancia 🔒

Aplicaciones como el reconocimiento facial masivo o la predicción de comportamiento pueden invadir la privacidad y generar sociedades de vigilancia.

Uso malicioso de IA

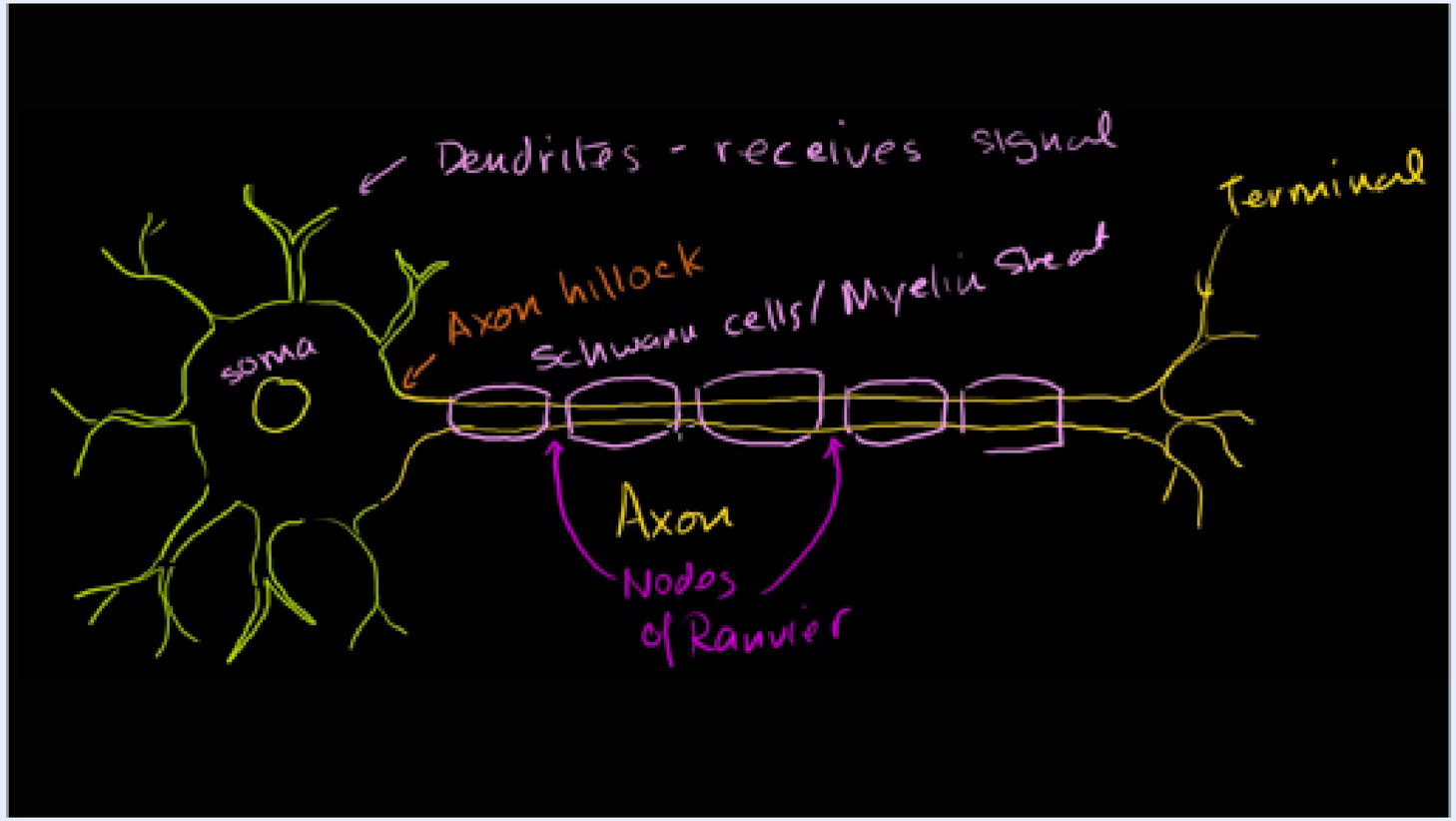
Desde la creación de deepfakes hasta la automatización de ciberataques, los modelos de deep learning pueden ser utilizados para fines dañinos o ilegales.

Redes neuronales artificiales

Surgieron como un intento de imitar el funcionamiento del cerebro humano. En el cerebro, las neuronas biológicas se conectan a través de sinapsis formando una compleja red

Una red biológica:

- Recibe señales químicas a través de las dendritas
- Procesa estas señales en el cuerpo celular (soma)
- Si la suma de señales supera cierto umbral, genera un impulso eléctrico
- Transmite este impulso a otras neuronas a través del axón



Funcionamiento

Una red neuronal artificial funciona como un sistema de procesamiento de información que

1. Recibe datos de entrada en forma de vectores o matrices
2. Transforma estos datos a través de operaciones matemáticas en cada capa
3. Produce una salida que representa la solución al problema (clasificación, predicción, etc.)

1. Multiplicación matricial (datos × pesos)
2. Adición de términos de sesgo
3. Aplicación de funciones no lineales (activación)

Redes neuronales convolucionales (CNN)

- Especializadas en procesamiento de imágenes y datos con estructura
- Detectan características locales como bordes, texturas y patrones
- Ejemplo: ResNet, VGG, Inception (usadas en reconocimiento facial, detección de objetos)

Redes neuronales recurrentes (RNN)

- Diseñadas para datos secuenciales (texto, audio, series temporales)
- Tienen "memoria" de entradas anteriores
- Variantes importantes: LSTM (Long Short-Term Memory) y GRU
- Aplicaciones: traducción automática, generación de texto

Componentes: capas, pesos, funciones de activación

Capa de entrada

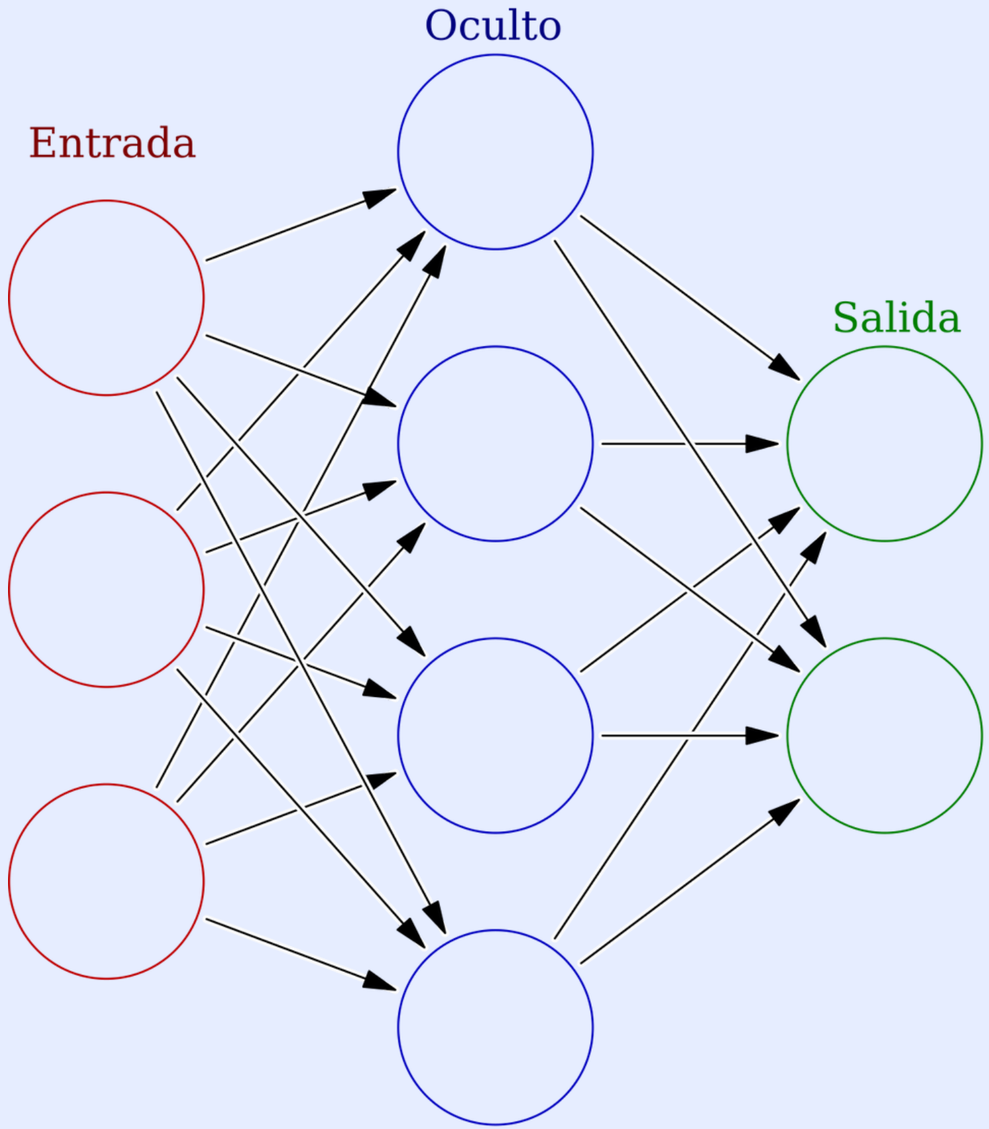
- Función: Recibe los datos originales del problema
- Características:
  - Tiene tantas neuronas como características (features) de entrada
  - No realiza cálculos, solo distribuye la información

Capas ocultas

- Función: Realizan la extracción y transformación de características
- Características:
  - Su número y tamaño determinan la capacidad de la red
  - Cada capa adicional permite aprender representaciones más abstractas
  - Las redes con muchas capas ocultas se denominan "profundas" (deep learning)

Capa de salida

- Función: Produce el resultado final de la red
- Características:
  - Su estructura depende del tipo de problema:
  - Clasificación binaria: 1 neurona con activación sigmoid
  - Regresión: 1 o más neuronas con activación lineal



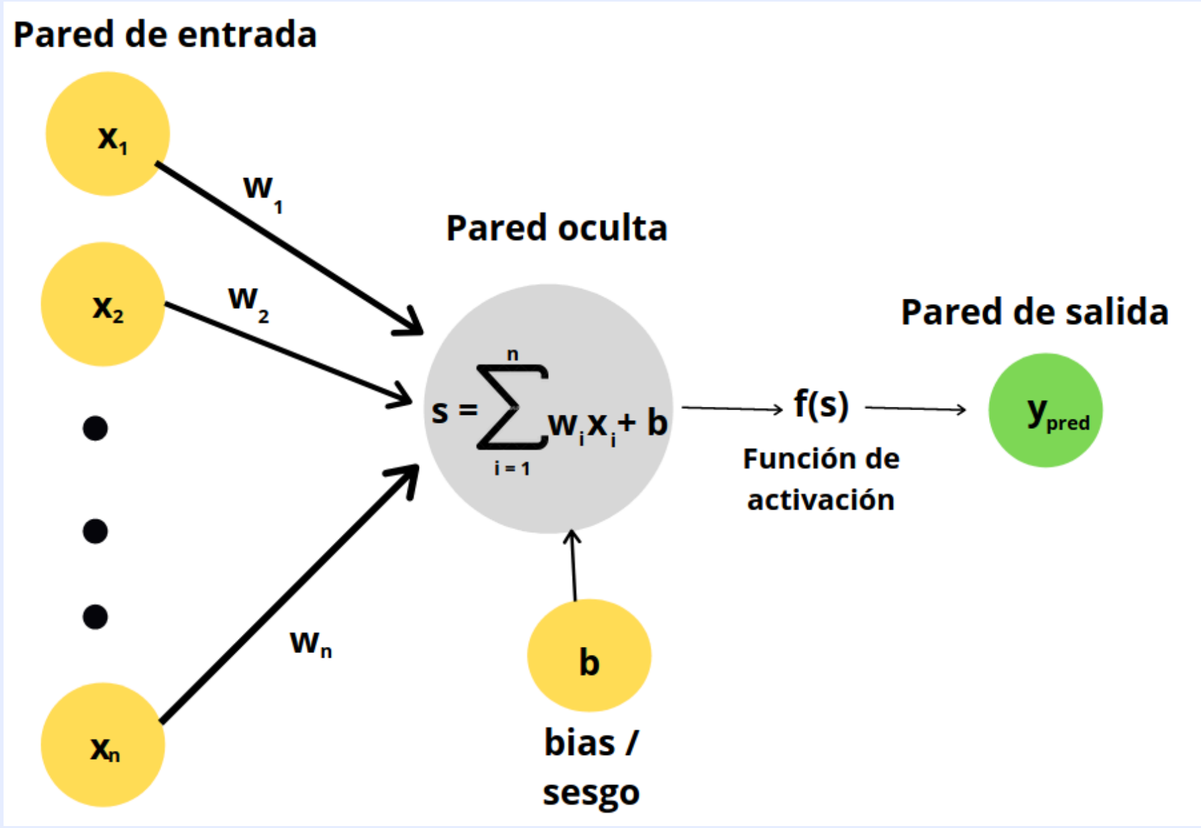
Pesos y sesgos

Pesos (weights)

- Definición: Valores numéricos que determinan la fuerza de conexión entre neuronas
- Características:
  - Son los parámetros principales que la red ajusta durante el entrenamiento
  - Valores grandes indican conexiones importantes
  - Valores cercanos a cero indican conexiones débiles
  - Pueden ser positivos (excitatorios) o negativos (inhibitorios)

Sesgos (bias)

- Definición: Valores constantes añadidos a la suma ponderada en cada neurona
- Características:
  - Permiten desplazar la función de activación horizontalmente
  - Añaden un grado de libertad adicional al modelo
  - Funcionan como un umbral de activación ajustable
  - Matemáticamente, equivalen a una entrada adicional



Principales funciones de activación

Sigmoid (Logística)

Fórmula: σ(x) = 1/(1+e^(-x))

Rango de salida: (0, 1)

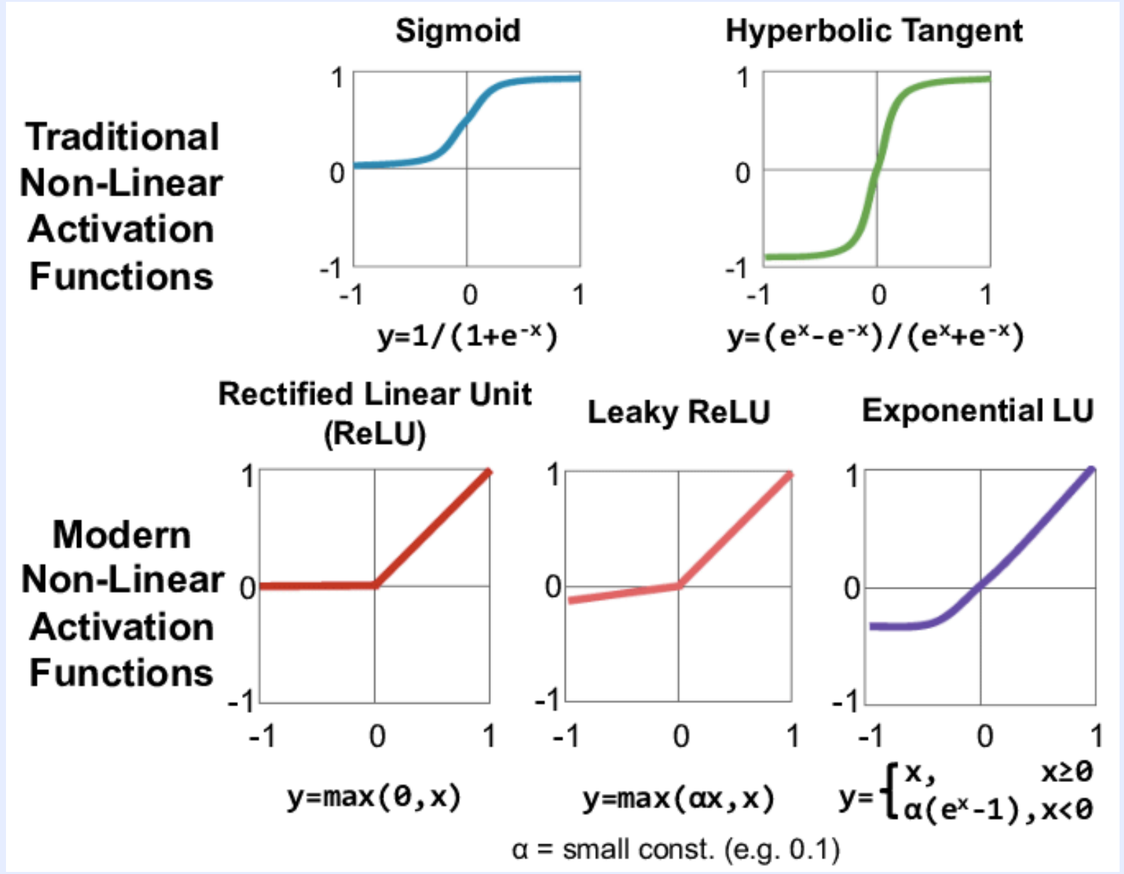
- Características:
  - Útil en la capa de salida para clasificación binaria
  - Problemas: desvanecimiento del gradiente en redes profundas, salidas no centradas en cero

Tanh (Tangente hiperbólica)

Fórmula: tanh(x) = (e^x - e^(-x))/(e^x + e^(-x))

Rango de salida: (-1, 1)

- Características:
  - Similar a sigmoid pero centrada en cero
  - Gradientes más pronunciados
  - Sufre también de desvanecimiento del gradiente



Proceso de entrenamiento: forward + backpropagation

Proceso mediante el cual se ajustan sus pesos y sesgos para minimizar una función de error o pérdida, se divide principalmente en dos fases que se repiten iterativamente:

Forward propagation (propagación hacia adelante):

1. Entrada de datos: Un lote de datos (batch) se proporciona a la capa de entrada.
2. Operaciones en cada capa:

Para cada neurona j en la capa l:

$z\_j^{(l)} = \Sigma(w\_ji^{(l)} * a\_i^{(l-1)}) + b\_j^{(l)}$

Donde:

- $z\_j^{(l)}$ es la entrada neta a la neurona
- $w\_ji^{(l)}$ es el peso de la conexión
- $a\_i^{(l-1)}$ es la activación de la neurona en la capa anterior
- $b\_j^{(l)}$ es el sesgo

3. Aplicación de la función de activación:

$a\_j^{(l)} = f(z\_j^{(l)})$ (f es la función de activación seleccionada)

4. Propagación secuencial: La salida de cada capa se convierte en la entrada de la siguiente

5. Salida final: La capa de salida genera la predicción $\hat{y}$ (y-hat)

Backpropagation (Retropropagación)

1. Cálculo del error: Se compara la predicción con el valor real.

$E = L(\hat{y}, y)$ (L es la función de pérdida (loss function))

2. Cálculo del gradiente en la capa de salida:

$\boldsymbol{\delta}^{(L)} = \nabla\_a L \odot f'(z^{(L)})$

- $\boldsymbol{\delta}^{(L)}$ es el error en la capa de salida
- $\nabla\_a L$ es el gradiente de la función de pérdida respecto a las activaciones
- f' es la derivada de la función de activación
- $\odot$ representa el producto elemento por elemento

3. Propagación del error hacia atrás: En cada capa se ajusta el error teniendo en cuenta cómo los pesos y activaciones anteriores influyeron en el resultado.

4. Cálculo de los gradientes: Modifica los pesos de forma que, si volvemos a pasar los mismos datos, el error disminuya.

5. Actualización de pesos y sesgos: Se le resta una pequeña parte del error que causó multiplicado por una tasa de aprendizaje.

Visualización del error durante el entrenamiento

El error (también llamado pérdida o "loss") es una medida de qué tan inexactas son las predicciones de la red neuronal en comparación con los valores reales esperados, una cuantificación de la diferencia entre lo que la red predice y la respuesta correcta

¿Por qué se produce?

- La red neuronal está en proceso de aprendizaje y aún no ha encontrado los pesos óptimos
- Los datos son complejos y la arquitectura de la red podría no ser suficiente
- Existe ruido inherente en los datos
- La red puede estar sobreajustando (overfitting) o subajustando (underfitting) los datos

Importancia de la visualización del error

- Diagnóstico de problemas: Permite identificar problemas como sobreajuste o subajuste
- Ajuste de hiperparámetros: Ayuda a determinar cuándo detener el entrenamiento y a ajustar hiperparámetros
- Evaluación del rendimiento: evaluación de qué tan bien está aprendiendo la red

Técnicas de visualización comunes

1. Gráficas de error por época: La visualización más básica y común

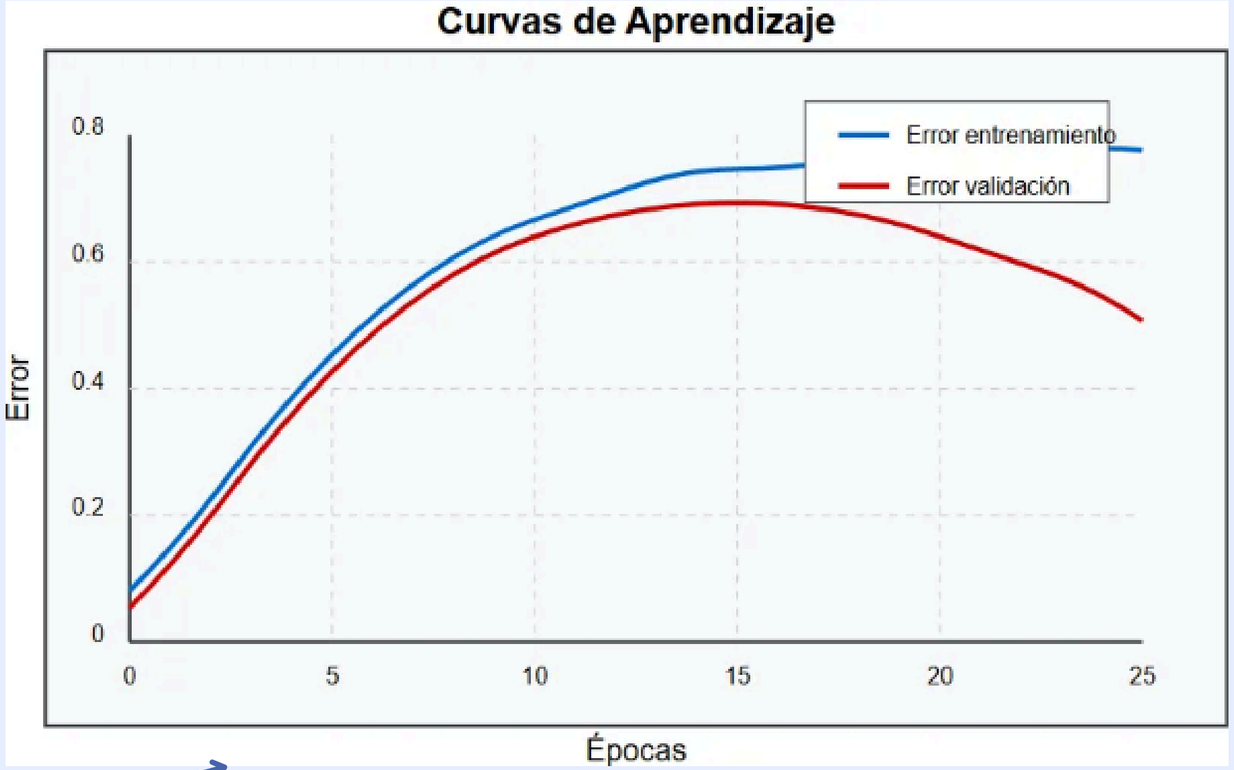
2. Comparación de error de entrenamiento vs validación: Fundamental para detectar sobreajuste

3.Gráficas de distribución del error: Muestra cómo se distribuye el error entre diferentes muestras

Gráficas de error: Curva de aprendizaje

El gráfico representa la curva que se obtiene durante el proceso de entrenamiento de la red neurona

- Cálculo del error: En cada época, la red:
  - Procesa todos los datos de entrenamiento (forward pass)
  - Calcula la diferencia entre sus predicciones y los valores reales (error)
  - Hace lo mismo con un conjunto separado de datos de validación
- Almacenamiento: Estos valores de error se guardan para cada época



Elementos

- Eje X (Épocas): número de iteraciones completas sobre todo el conjunto de datos
- Eje Y (Error): Muestra la magnitud del error o pérdida
- Línea azul: Representa el error en los datos de entrenamiento
- Línea roja: Representa el error en los datos de validación (datos que la red no ha visto durante el entrenamiento)

Fase inicial (épocas 0-15):

- Ambos errores aumentan y podría indicar que la red está "desaprendiendo" patrones iniciales o que hay una configuración de entrenamiento particular

Fase posterior (épocas 15-25):

- El error de entrenamiento (azul) se estabiliza
- El error de validación (rojo) comienza a disminuir

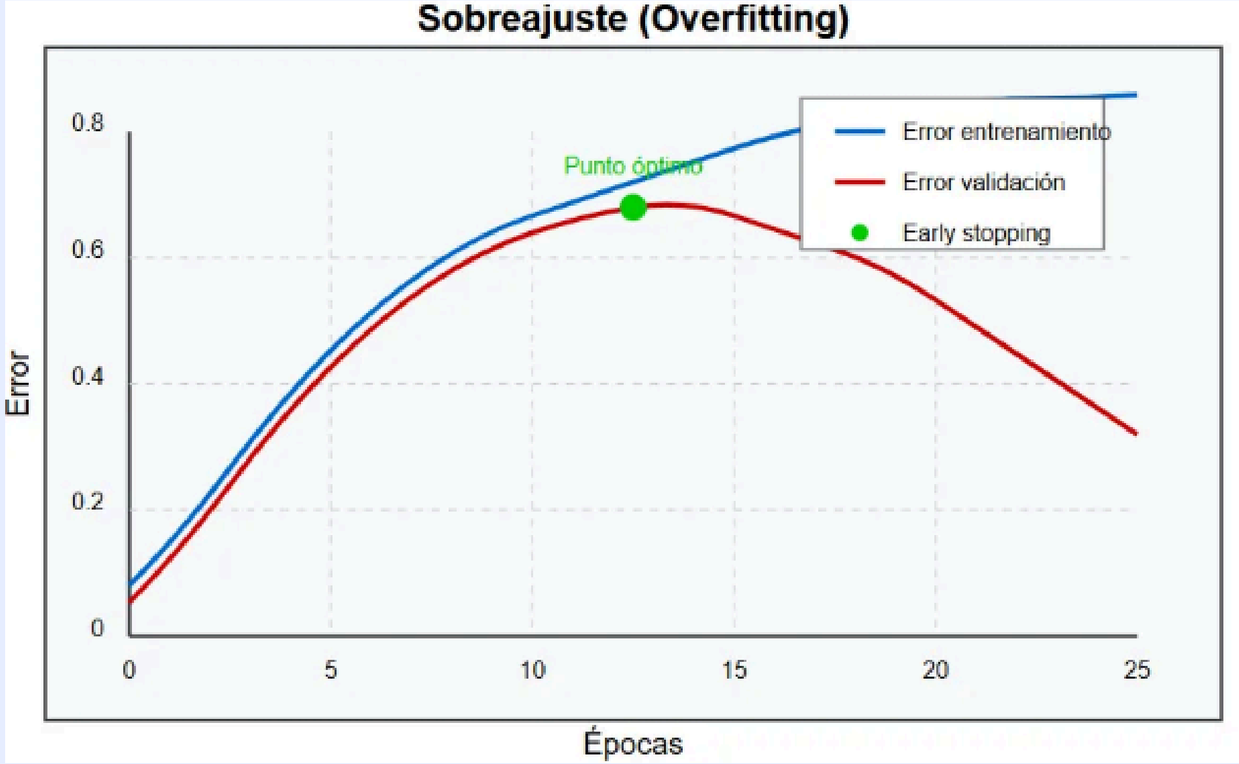
Gráficas de error: Sobreajuste

1.Fase inicial (épocas 0-12):

- Ambas curvas (entrenamiento y validación) aumentan juntas
- La red está aprendiendo patrones útiles que generalizan bien

3. Fase de sobreajuste (épocas 13-25):

- Error de entrenamiento (azul): Continúa aumentando (incluso llega casi a 0.85)
- Error de validación (rojo): Comienza a disminuir significativamente
- Esta divergencia es la señal clara de overfitting



Después del punto óptimo, la red neuronal:

- Sigue generalizando datos nuevos (por eso el error de validación cae)
- Pero pierde capacidad de "memorizar" los datos de entrenamiento (por eso el error de entrenamiento aumenta)
- La red se está especializando en solo ser receptor de datos

2. Punto crítico (época ~12-13):

- Este es el punto óptimo donde el modelo generaliza mejor

Early Stopping (Punto verde):

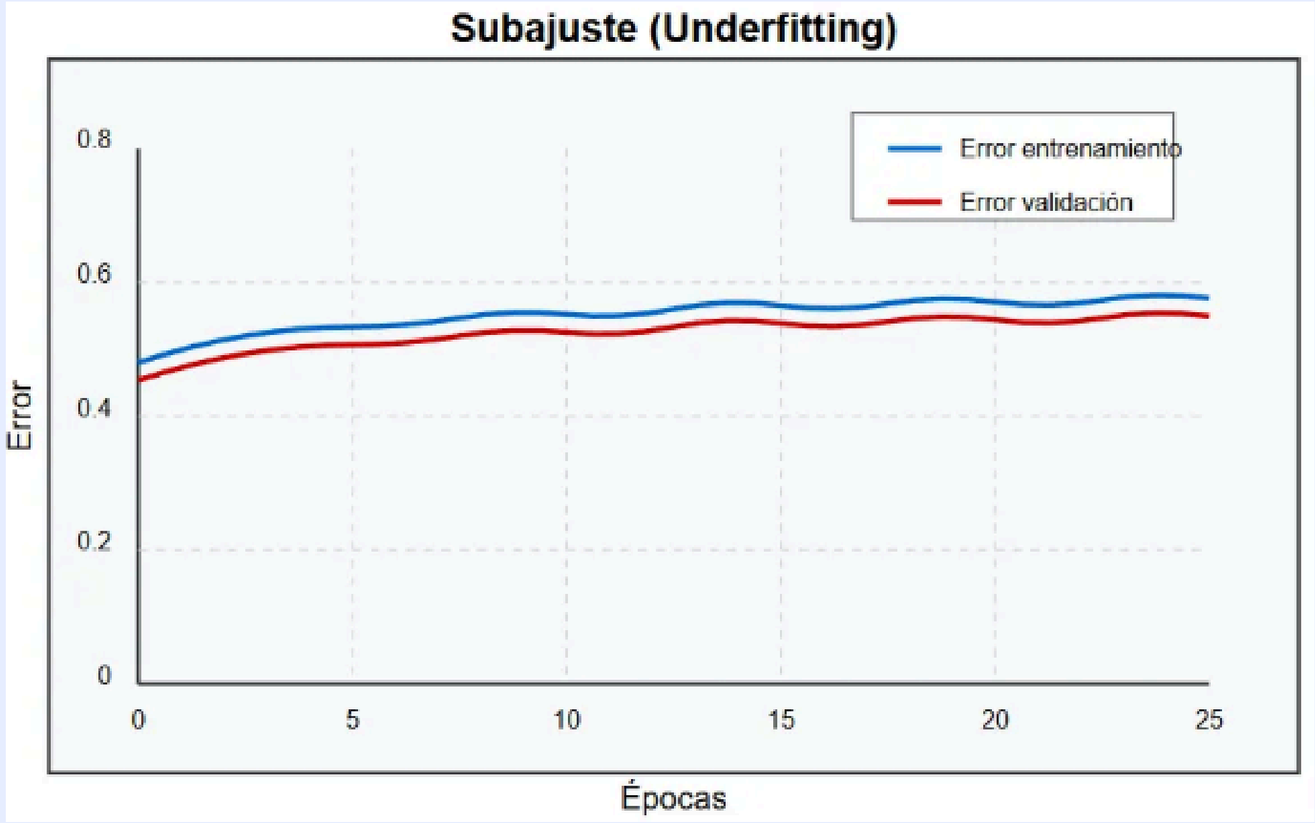
- Es donde deberíamos haber detenido el entrenamiento
- Representa el mejor balance entre aprender patrones útiles y mantener la capacidad de generalización
- Entrenar más allá de este punto solo empeora el rendimiento del modelo

Gráficas de error: Subajuste

Características:

1. Errores altos y estables
   - Tanto el error de entrenamiento (azul) como el de validación (rojo) se mantienen altos (alrededor de 0.5-0.55)
   - No hay una mejora significativa a lo largo de las 25 épocas

2. Falta de aprendizaje
   - Las curvas son prácticamente planas
   - La red no está captando los patrones en los datos
   - El modelo es demasiado simple para la complejidad del problema



3. Curvas paralelas
   - Ambas curvas están muy próximas entre sí
   - No hay divergencia como en el overfitting

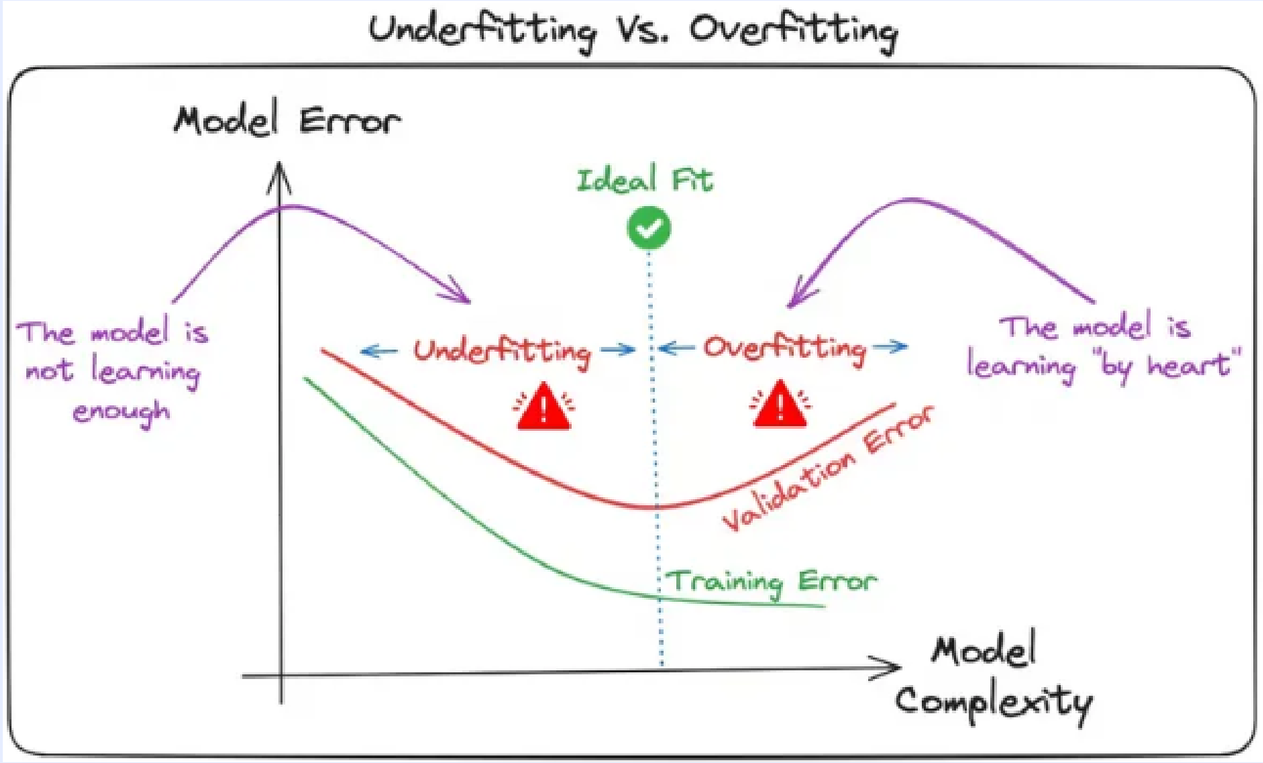
Indica que el modelo tiene un rendimiento similar en datos de entrenamiento y validación, pero ambos son pobres

Balance entre underfitting y overfitting

La gráfica nos muestra que existe un punto óptimo de complejidad del modelo. Ni muy simple, ni muy complejo

1. Zona de Underfitting (Izquierda)

- El training error es alto (curva verde alta)
- El validation error también es alto (curva roja alta)
- Ambas curvas están cerca → el modelo "no está aprendiendo lo suficiente"
- Es como darle un problema de cálculo a un niño de primaria



Elementos

Ejes:

- Eje X (Model Complexity): Representa qué tan complejo es el modelo (pocas neuronas/capas → muchas neuronas/capas)
- Eje Y (Model Error): Representa la magnitud del error del modelo

Curvas:

- Curva verde (Training Error): Error en datos de entrenamiento
- Curva roja (Validation Error): Error en datos de validación

2. Punto Ideal (Centro)

- Marcado con la ✓ verde y "Ideal Fit"
- Ambos errores están en su punto más bajo
- Las curvas están próximas pero el modelo sí está aprendiendo
- Este es nuestro objetivo: generaliza bien sin memorizar

3. Zona de Overfitting (Derecha)

- El training error continúa bajando
- El validation error comienza a subir
- El modelo está "aprendiendo de memoria"
- Como memorizar las respuestas pero no entender el concepto

Muchas Gracias