



# **Inteligencia Artificial Aplicada para la Economía**



## **Profesores**

**Profesor Magistral**

Camilo Vega Barbosa

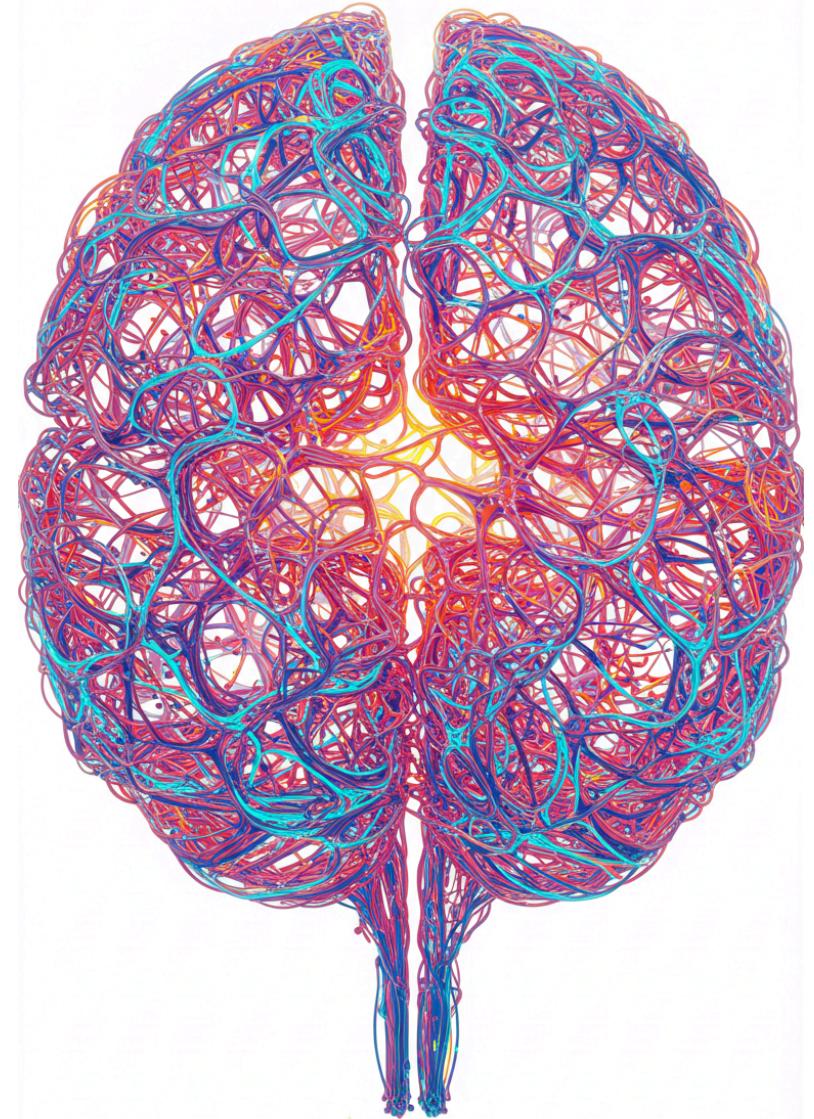
**Asistente de Docencia**

Sergio Julian Zona Moreno



# Deep Learning

## Fundamentos de las Redes Neuronales Profundas



# ¿Qué es el Deep Learning?

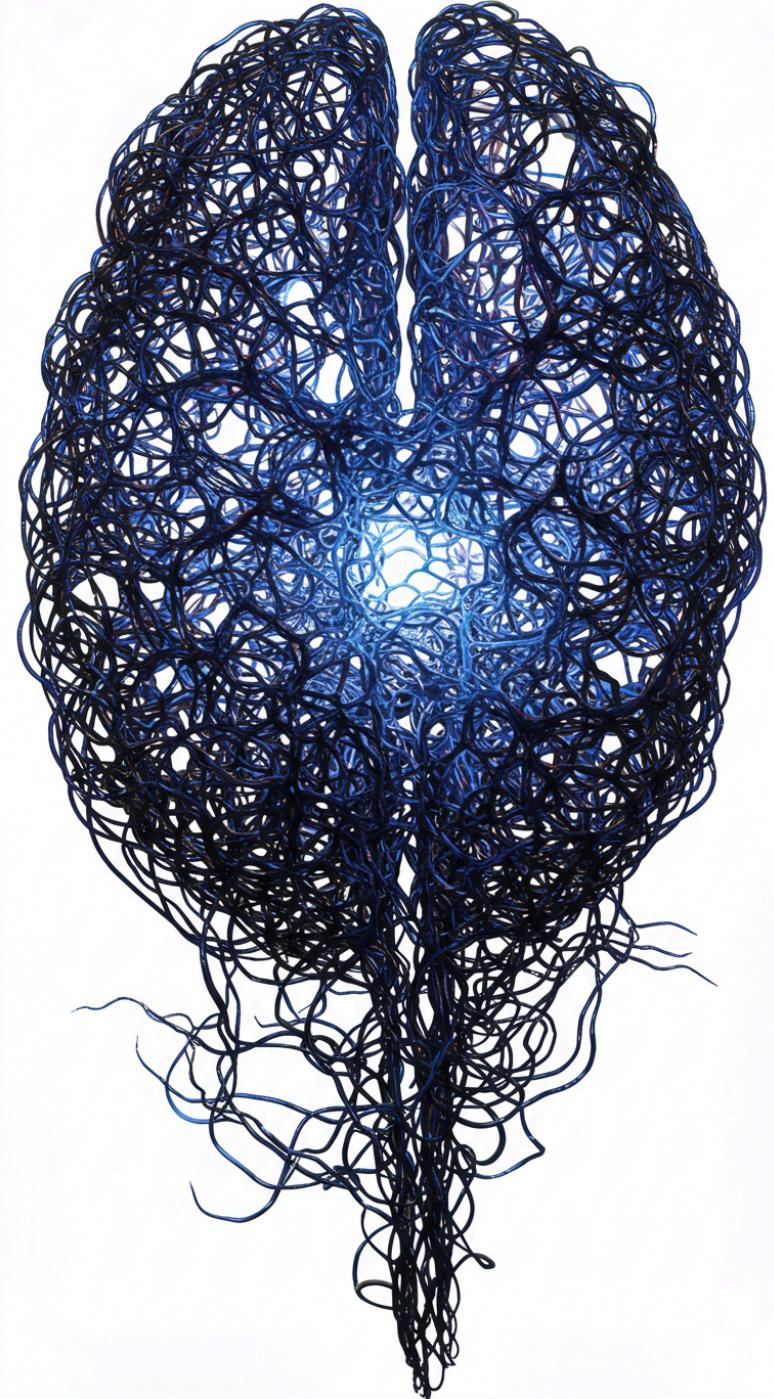
El Deep Learning es una rama del Machine Learning que usa **redes neuronales profundas** para aprender representaciones complejas de datos. Funciona en aprendizaje **supervisado** (datos etiquetados) y **no supervisado** (sin etiquetas).

A diferencia de métodos tradicionales, extrae patrones automáticamente mediante **entrenamiento iterativo y backpropagation**, ajustando millones de parámetros para mejorar el rendimiento.

## 🎯 Aplicaciones en Economía

-  Predicción financiera y detección de anomalías
-  Evaluación de riesgo crediticio
-  Análisis de tendencias macroeconómicas

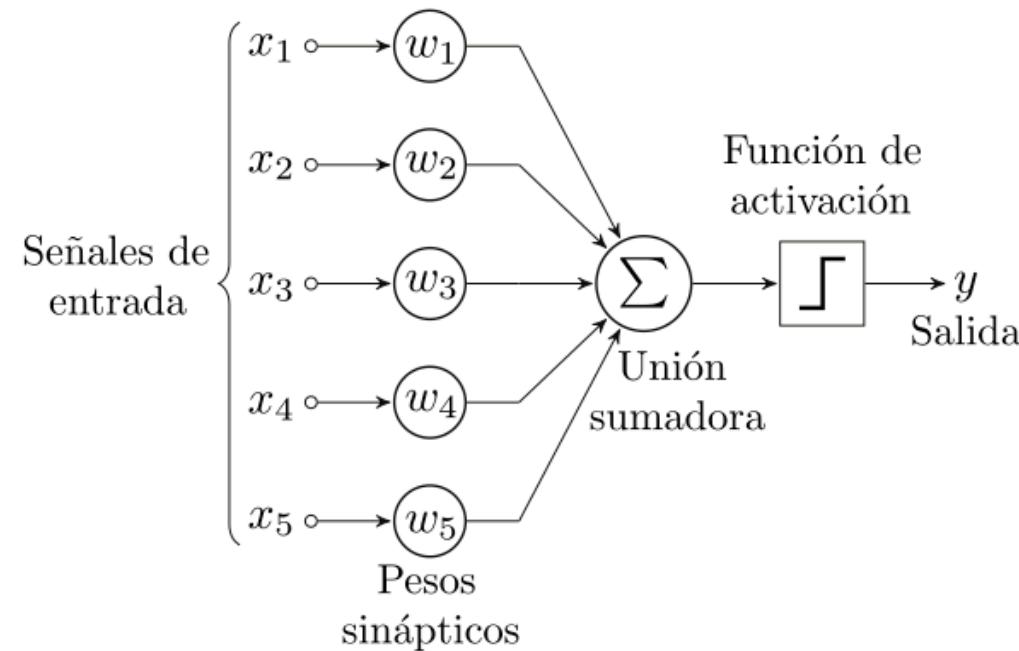
# La primera red neuronal



# El Perceptrón: Inspirado en el Cerebro Humano

El perceptrón, creado por **Frank Rosenblatt** en 1957, fue el primer modelo de **red neuronal artificial**, inspirado en el funcionamiento de las **neuronas biológicas**.

Rosenblatt buscaba replicar cómo las neuronas procesan información en el cerebro: **recibiendo señales, ponderándolas según su importancia y produciendo una respuesta cuando se alcanza cierto umbral de activación.**

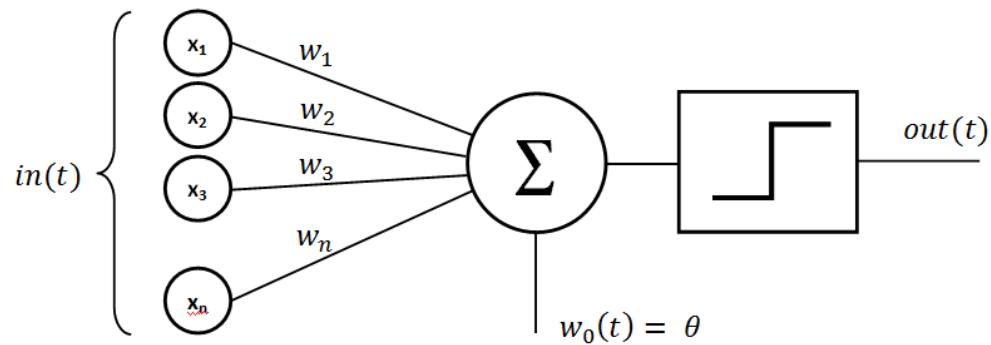




# Anatomía del Perceptrón

## Entradas y Pesos

- **Entradas ( $x$ ):** Señales numéricas que ingresan a la neurona.
- **Pesos ( $w$ ):** Determinan la influencia de cada entrada.
- **Umbral ( $\theta$ ):** Define el punto de activación de la neurona. También se suele llamar *bias*.

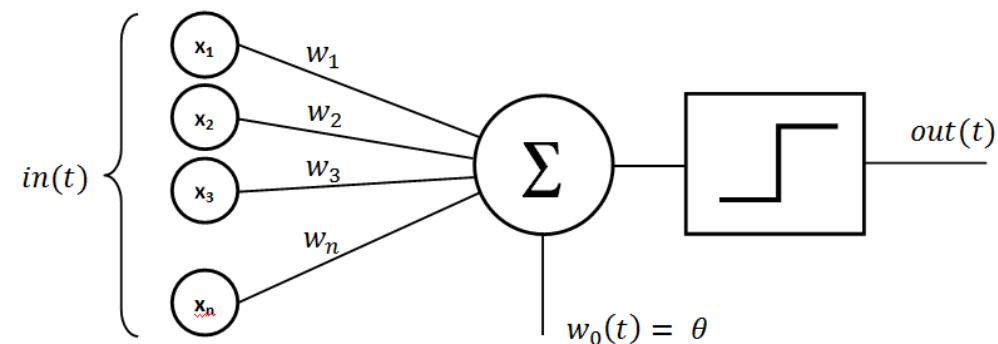




# Anatomía del Perceptrón

## ⚡ Procesamiento

- 1 Suma Ponderada:** Calcula la combinación lineal de entradas y pesos.
- 2 Función de Activación:** Decide si la neurona se activa.
- 3 Salida ( $y$ ):** Genera un resultado binario (0 o 1) según el umbral.





# Anatomía del Perceptrón

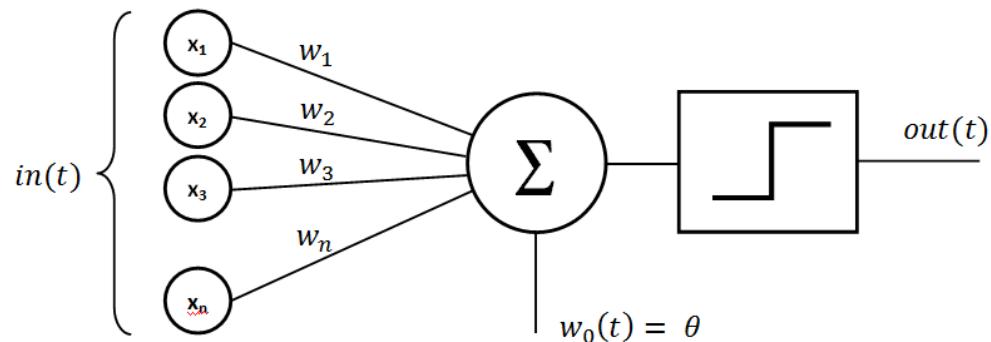


## Fórmula Matemática

$$y = f(w_1x_1 + w_2x_2 + \dots + w_nx_n - \theta)$$

Donde  $f$  es la función de activación (Escalón):

$$f(z) = \begin{cases} 1, & \text{si } z > 0 \\ 0, & \text{en otro caso} \end{cases}$$





## Ejemplo de cálculo en un Perceptrón Básico

Dado un perceptrón con **dos entradas** y una salida binaria, queremos calcular su activación:

- Entradas:  $x_1, x_2$
- Pesos:  $w_1, w_2$
- Umbral:  $\theta$
- Función de activación: Escalón unitario



## Solución del Perceptrón:

### 1 Datos

- Pesos:  $w_1 = 0.6, w_2 = -0.8, \theta = 0.2$
- Entrada:  $x_1 = 1, x_2 = 0$

### 2 Fórmula General

$$z = w_1x_1 + w_2x_2 - \theta$$

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

### 3 Cálculo

$$z = (0.6)(1) + (-0.8)(0) - 0.2 = 0.6 - 0.2 = 0.4$$

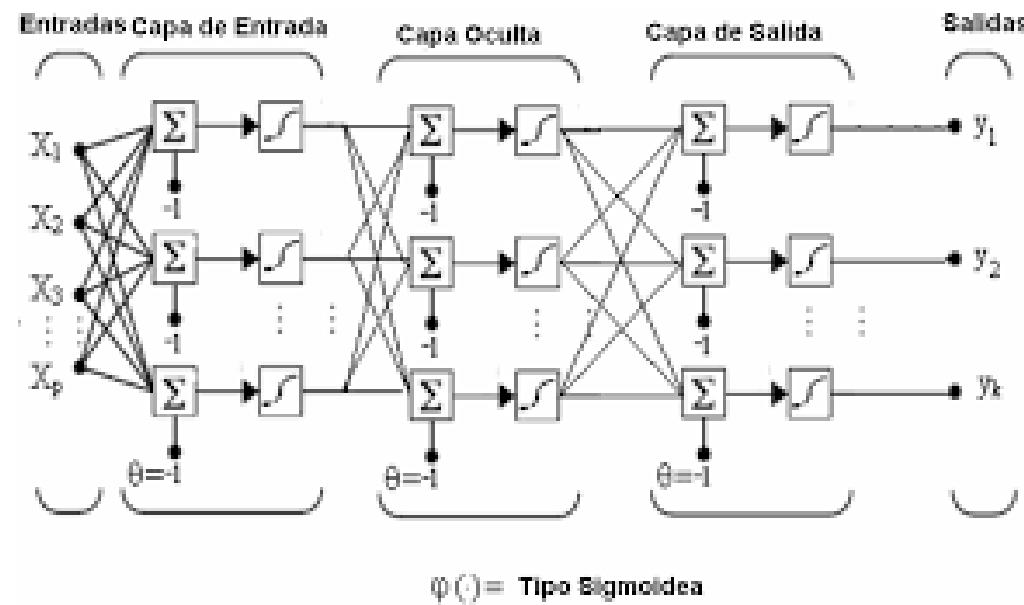
Como  $z > 0$ , la neurona se activa y la salida es  $y = 1$

# Perceptrón Multicapa (MLP)

El perceptrón multicapa extiende el concepto básico agregando **capas intermedias** (ocultas) entre la entrada y la salida. Esto permite aprender representaciones más complejas.

## ⟳ Estructura del MLP

- **Capa de Entrada:** Recibe los datos originales
- **Capas Ocultas:** Procesan y transforman la información
- **Capa de Salida:** Produce el resultado final



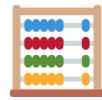


# El Proceso Feed Forward

## ⚡ ¿Cómo Funciona?

El término "feed forward" significa que la información fluye hacia adelante, desde la entrada hasta la salida, sin retroalimentación:

- 1 Los datos entran por la primera capa
- 2 Cada neurona calcula su activación y la pasa a la siguiente capa
- 3 La información solo avanza, nunca retrocede
- 4 La última capa produce la predicción final



# Matemática del MLP

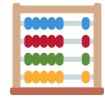
## 💡 Cálculo en Cada Neurona

Para cada neurona en cualquier capa, el cálculo es:

$$y = f\left(\sum_{i=1}^n w_i x_i + b\right)$$

Donde ahora:

- $f$  es una función de activación no lineal (ReLU, sigmoid, tanh)
- $b$  es el bias (antes  $\theta$ )
- La salida  $y$  es continua, no binaria



# Matemática del MLP



## Funciones de Activación Comunes

$$\text{ReLU}(x) = \max(0, x)$$

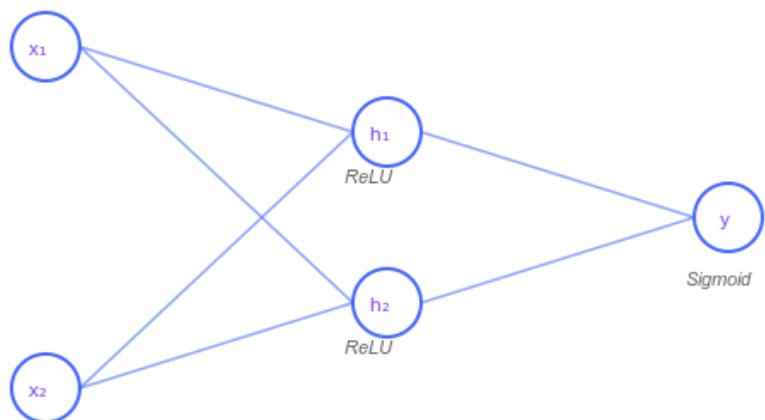
$$\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

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



# Ejemplo de MLP

Capa de Entrada      Capa Oculta (ReLU)      Capa de Salida (Sigmoid)



## Estructura de la Red

- Entrada: 2 neuronas ( $x_1, x_2$ )
- Capa Oculta: 2 neuronas con ReLU
- Salida: 1 neurona con sigmoid



## Datos de Entrada

$$x = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$



# Parámetros de la Red

## 12 34 Capa Oculta

$$W_1 = \begin{bmatrix} 0.5 & -0.2 \\ 0.1 & 0.8 \end{bmatrix}$$

$$b_1 = \begin{bmatrix} 0.3 \\ -0.4 \end{bmatrix}$$

## Capa de Salida

$$W_2 = \begin{bmatrix} 0.7 & 0.3 \end{bmatrix}$$

$$b_2 = 0.2$$



# Cálculo en la Red Neural

## 1 Capa Oculta (ReLU)

$$\begin{aligned} h &= \text{ReLU}(W_1x + b_1) \\ &= \text{ReLU} \left( \begin{bmatrix} 0.5 & -0.2 \\ 0.1 & 0.8 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 0.3 \\ -0.4 \end{bmatrix} \right) \\ &= \text{ReLU} \begin{bmatrix} 0.1 \\ 1.3 \end{bmatrix} = \begin{bmatrix} 0.1 \\ 1.3 \end{bmatrix} \end{aligned}$$



# Cálculo en la Red Neural

## 2 Capa de Salida (Sigmoid)

$$\begin{aligned}y &= \sigma(W_2 h + b_2) \\&= \sigma((0.7 \cdot 0.1 + 0.3 \cdot 1.3) + 0.2) \\&= \sigma(0.07 + 0.39 + 0.2) = \sigma(0.66) \\&= 0.659\end{aligned}$$

## 🔍 Conclusión

- La red se activó con una confianza del 65.9%
- Este valor sugiere una predicción positiva moderada

# El inicio del Deep Learning



# Del MLP al Deep Learning 🚀

El Deep Learning revoluciona las redes neuronales al utilizar **4 o más capas ocultas** 🧠, superando la estructura básica del MLP tradicional. Esta profundidad permite que la red procese información en niveles cada vez más abstractos ⚡, transformando su capacidad de aprendizaje.

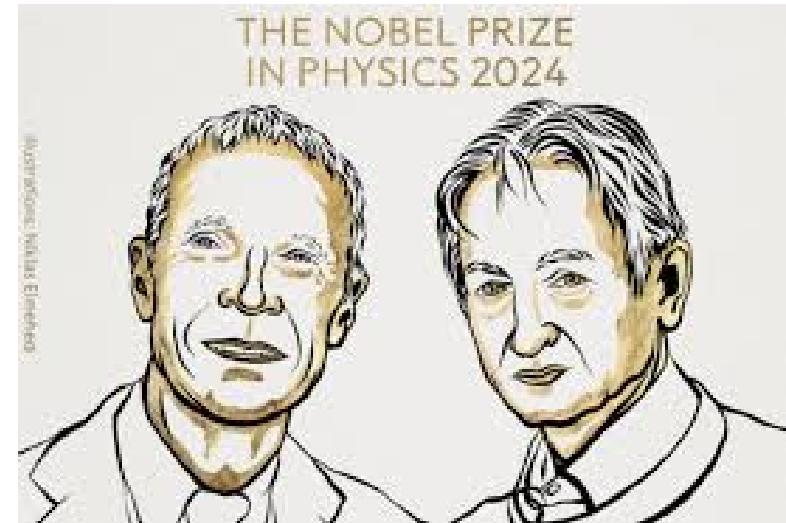
Cada capa construye sobre las anteriores de manera progresiva 🎯: desde **características básicas** hasta **patrones complejos** 🔍, permitiendo abordar tareas sofisticadas como **visión por computador**, **procesamiento de lenguaje** y **análisis financiero** 💰.

# Los pioneros del Deep Learning



Los años 80 sentaron las bases teóricas que harían posible el Deep Learning moderno:

- 🎯 1982: **John Hopfield** revoluciona el campo introduciendo las **redes recurrentes**, demostrando que las redes neuronales podían tener memoria y procesar secuencias temporales
- 💫 1986: **Geoffrey Hinton** desarrolla el algoritmo de **backpropagation**, resolviendo el problema fundamental de cómo entrenar redes profundas de manera eficiente





# Backpropagation: El Motor del Aprendizaje

El **backpropagation** es el algoritmo que permite a las redes neuronales **aprender** de sus errores, ajustando sus pesos para mejorar sus predicciones.

## 🎯 Conceptos Clave

- Calcula cómo cada peso contribuye al error total
- Propaga el error desde la salida hacia atrás
- Actualiza los pesos para minimizar el error



# Matemática del Backpropagation

**1** Forward Pass: Calcular la predicción y el error

$$E = \frac{1}{2}(y_{real} - y_{pred})^2$$

**2** Backward Pass: Calcular gradientes

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_{pred}} \cdot \frac{\partial y_{pred}}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_{ij}}$$

**3** Actualización: Ajustar pesos

$$w_{nuevo} = w_{viejo} - \eta \frac{\partial E}{\partial w}$$

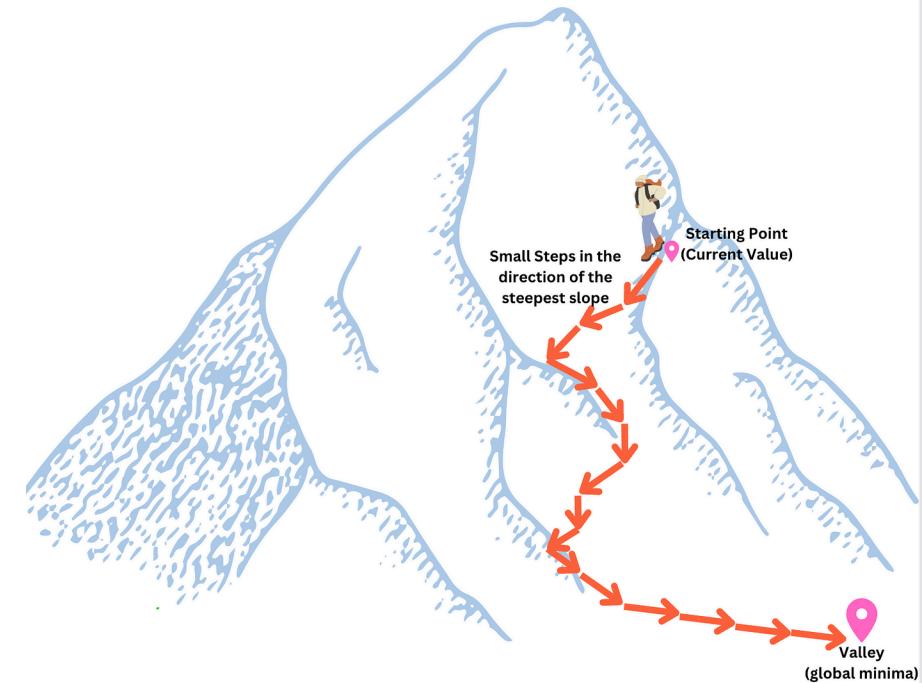
Donde  $\eta$  es la tasa de aprendizaje (learning rate)

# La Caída del Gradiente



Imagina que el error de nuestra red neuronal es como una montaña con muchos valles 🌫️. Cuanto más alto estamos, más grande es el error. Nuestro objetivo es llegar al punto más bajo posible, donde el error sea mínimo.

La **caída del gradiente** funciona como una pelota que soltamos en esta montaña ⚡: naturalmente rodará hacia abajo, tomando el camino más empinado en cada punto. El **learning rate** determina qué tan rápido rueda esta pelota: si va muy rápido podría saltar el valle más profundo, si va muy lento tardará demasiado en llegar.



# Backpropagation y Gradiente: El Dúo del Aprendizaje



El **backpropagation** y la **caída del gradiente** trabajan juntos como un equipo perfecto en el Deep Learning 🤝. Mientras el backpropagation calcula cómo cada neurona contribuyó al error, propagando esta información desde la última capa hasta la primera, la caída del gradiente usa esta información para ajustar los pesos en la dirección correcta.

## La Fórmula del Éxito 📈

1. **Backpropagation:** Calcula  $\partial E / \partial w$  (qué tanto afecta cada peso al error)
2. **Gradiente:** Ajusta  $w = w - \eta \partial E / \partial w$  (actualiza los pesos)
3. **Repetir:** Itera hasta encontrar los pesos óptimos

# Problemas y soluciones para entrenar redes neuronales





# Desafíos en el Entrenamiento

El entrenamiento de redes neuronales profundas presenta desafíos únicos que pueden afectar significativamente su rendimiento. El más crítico es el problema del **gradiente desvaneciente**.



## Vanishing Gradient

Los gradientes son la clave del aprendizaje en redes profundas. Cuando una red tiene muchas capas, estos gradientes pueden volverse extremadamente pequeños, causando que:

- Las capas más cercanas a la entrada **dejan de aprender**
- El entrenamiento se **estanca**
- La red no logra capturar patrones complejos



# Desafíos en el Entrenamiento

## Exploding Gradient

Mientras que el vanishing gradient hace que el aprendizaje se detenga, el exploding gradient causa el efecto contrario. Los valores crecen exponencialmente, generando un efecto dominó que:

- Causa **inestabilidad** en el entrenamiento
- Puede llevar a **desbordamiento numérico**
- Resulta en **pesos no óptimos**



# Soluciones a los Problemas de Gradientes

A lo largo de los años, la comunidad de deep learning ha desarrollado técnicas innovadoras para abordar estos desafíos. Cada solución ataca el problema desde un ángulo diferente.



## Técnicas Sugeridas

Las herramientas más efectivas en nuestro arsenal incluyen:

- **ReLU**: Evita la saturación que causa el vanishing gradient
- **Batch Normalization**: Mantiene las señales fuertes y estables
- **Skip Connections**: Crean autopistas para la información



# El Problema del Overfitting

Imagina un estudiante que memoriza perfectamente las respuestas de exámenes anteriores, pero no puede resolver problemas nuevos. Así funciona el **overfitting**: el modelo se vuelve demasiado específico a los datos de entrenamiento, perdiendo su capacidad de generalización.



## Señales de Overfitting

Las señales de advertencia suelen ser claras y consistentes:

- Rendimiento perfecto en entrenamiento
- Pobre generalización en validación
- Predicciones "demasiado confiadas"



## Estrategias de Prevención del Overfitting

La prevención del overfitting es un arte que combina diferentes técnicas. Como un sistema de control y balance, cada estrategia cumple un papel específico:

- El **Dropout** "apaga" neuronas aleatoriamente durante el entrenamiento, forzando a la red a ser más robusta.
- La **Data Augmentation** expone el modelo a más variaciones de los datos.
- El **Early Stopping** nos ayuda a encontrar el punto óptimo de generalización.



# Regularización

La regularización modifica la función de pérdida de la red neuronal añadiendo términos que penalizan la complejidad del modelo. Matemáticamente, si  $L$  es nuestra pérdida original, la regularización añade un término adicional:

$$L_{reg} = L + \lambda R(w)$$



## Tipos Principales

- **L1 (Lasso)**  $\lambda \sum |w_i|$ : Promueve la esparcidad, llevando algunos pesos a cero
- **L2 (Ridge)**  $\lambda \sum w_i^2$ : Penaliza pesos grandes de manera uniforme
- **ElasticNet**  $\lambda_1 \sum |w_i| + \lambda_2 \sum w_i^2$ : Combina las ventajas de L1 y L2

El parámetro  $\lambda$  controla la fuerza de la regularización.



# Hiperparámetros: El Arte del Ajuste

Como un chef ajustando los ingredientes de una receta, el diseño de redes neuronales requiere un delicado balance de hiperparámetros. Su correcta configuración determina el éxito del modelo.



## Hiperparámetros Críticos

Los parámetros fundamentales que debemos considerar son:

- **Learning Rate:** Controla la velocidad de aprendizaje
- **Batch Size:** Balancea velocidad y estabilidad
- **Arquitectura:** Define la capacidad del modelo
- **Regularización:** Previene el sobreajuste

# Métodos de Selección de Hiperparámetros

La selección de hiperparámetros es un proceso sistemático que requiere un enfoque metodológico riguroso. Existen varios métodos establecidos, cada uno con características distintivas.

## Estrategias Principales

El enfoque manual requiere experiencia previa y comprensión profunda del modelo. Se basa en ajustes iterativos fundamentados en resultados empíricos.

El **Grid Search** implementa una búsqueda sistemática evaluando todas las combinaciones posibles de hiperparámetros en un espacio discreto predefinido. Este método garantiza encontrar el óptimo global dentro del espacio de búsqueda especificado.



# Métodos de Selección de Hiperparámetros

El **Random Search** proporciona una alternativa eficiente al grid search, especialmente en espacios de alta dimensionalidad:

- Explora el espacio de manera más diversa
- Puede encontrar soluciones óptimas con menos evaluaciones
- Requiere menos recursos computacionales

La **Optimización Bayesiana** utiliza un enfoque probabilístico:

- Construye un modelo del espacio de hiperparámetros
- Optimiza basándose en resultados anteriores
- Especialmente efectiva para espacios complejos

# Recursos del Curso

## Plataformas y Enlaces Principales

 GitHub del curso

 [github.com/CamiloVga/IA\\_Aplicada](https://github.com/CamiloVga/IA_Aplicada)

 Asistente IA para el curso

 [Google Notebook LLM](#)