

# Machine learning y deep learning

Rolando Gonzales Martinez, PhD

Fellow postdoctoral Marie  
Skłodowska-Curie

Universidad de Groningen  
(Países Bajos)

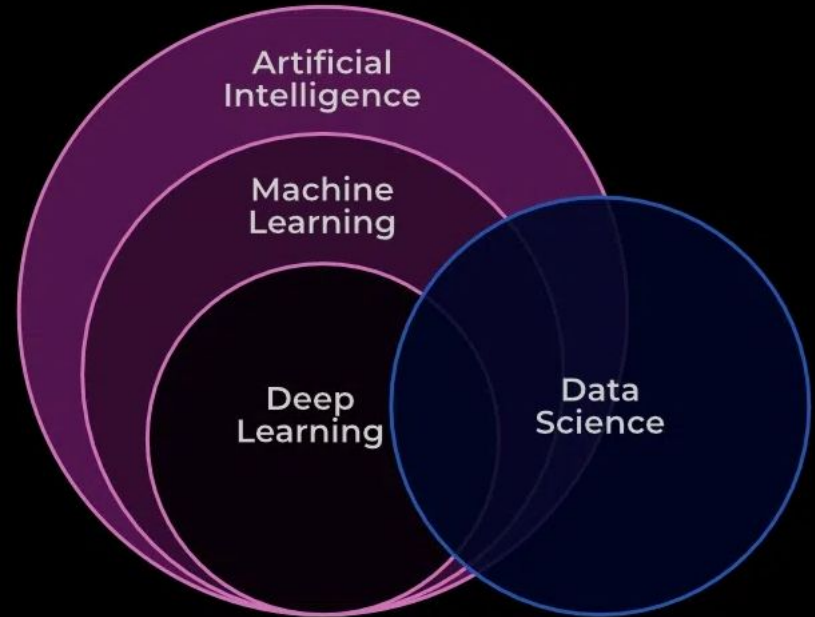
Investigador (researcher)

Iniciativa de Pobreza y Desarrollo  
Humano de la Universidad de  
Oxford (UK)

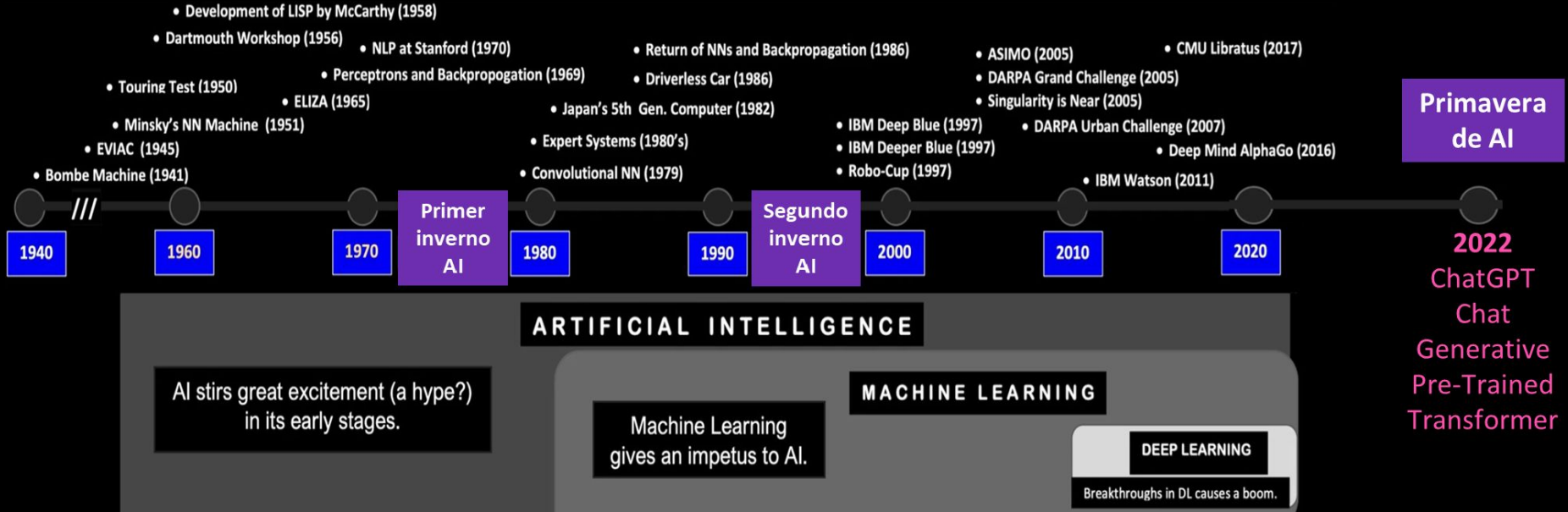
# Contenido del curso

## (5) Introducción al Deep Learning

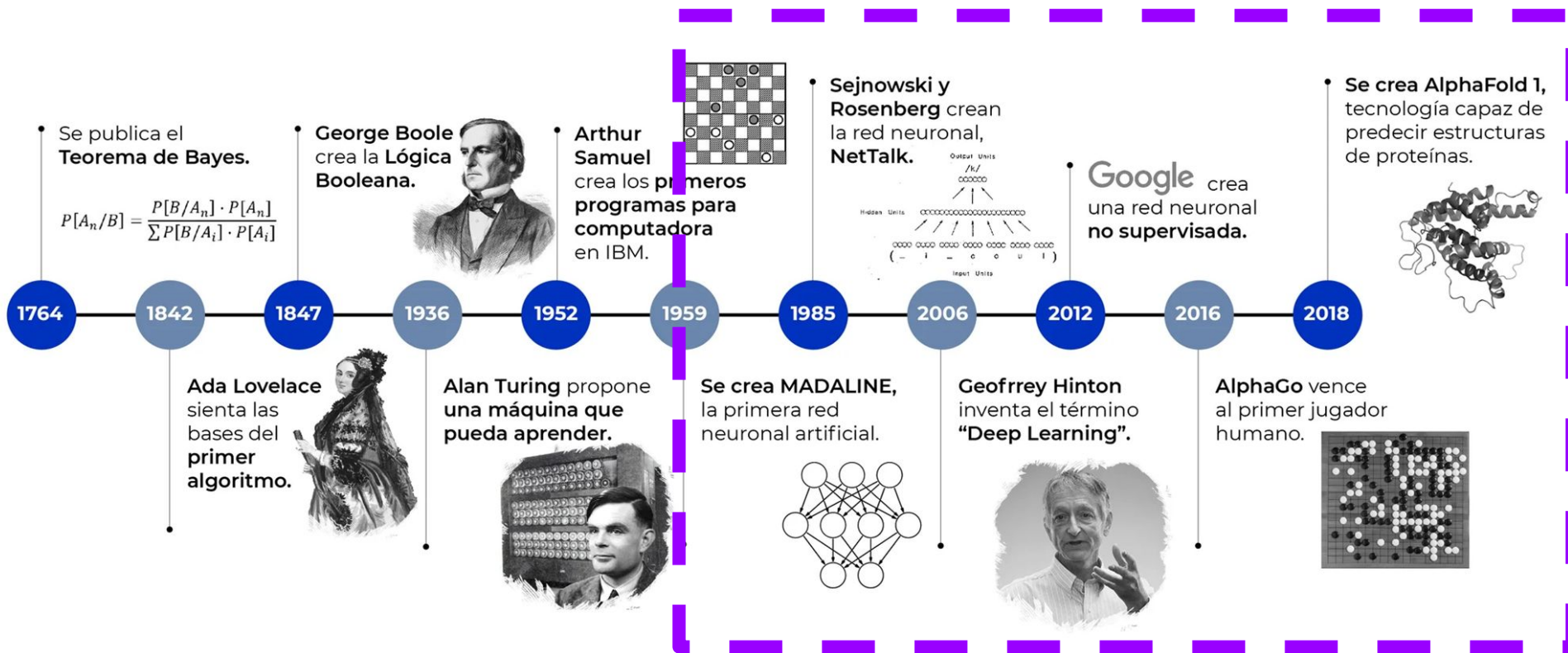
- Redes neuronales: perceptrones, redes multicapa.
- Funciones de activación, propagación hacia adelante y retropropagación.
- Laboratorio: Construcción de una red neuronal profunda con Keras.



# Línea de tiempo de la IA



# LÍNEA DEL TIEMPO DE MACHINE LEARNING



# Redes neuronales artificiales

Dada una función con un vector de entrada  $\mathbf{x}$  de dimensión  $k$ , y  $\theta$  parámetros del modelo, que incluye los pesos sinápticos y sesgos, una **red neuronal artificial** (RNA) es un modelo computacional parametrizado basado en una sucesión de funciones compuestas:

- En la función compuesta,  $L$  es el número de capas (layers)  
 $l = 1, 2, \dots, L$
- En las funciones de cada capa  $f(\mathbf{z}, \theta)$ ,  $\mathbf{z}$  es un vector de entrada que se obtiene de la salida de la capa anterior
- Las funciones de cada capa  $f(\mathbf{z}, \theta)$  son **expresiones lineales** en una función de activación **no lineal**  $\sigma(\cdot)$ , con pesos  $\mathbf{W}$  y sesgos  $\mathbf{b}$  para cada capa

$$f(\mathbf{x}; \theta) = f_L(f_{L-1}(\dots f_2(f_1(\mathbf{x}; \theta_1); \theta_2) \dots); \theta_L)$$

$$f_l(\mathbf{z}; \theta_l) = \sigma_l(\mathbf{W}_l \mathbf{z} + \mathbf{b}_l)$$

$$\theta = \{\mathbf{W}_l, \mathbf{b}_l\}_{l=1}^L$$

$$\mathcal{L}(f(\mathbf{x}; \theta), \mathbf{y})$$

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_{\theta} \mathcal{L}(f(\mathbf{x}; \theta), \mathbf{y})$$

$$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial E}{\partial w_{ij}}$$

# Redes neuronales artificiales

Dada una función con un vector de entrada  $\mathbf{x}$  de dimensión  $k$ , y  $\theta$  parámetros del modelo, que incluye los pesos sinápticos y sesgos, una **red neuronal artificial** (RNA) es un modelo computacional parametrizado basado en una sucesión de funciones compuestas:

- La red neuronal se entrena ajustando los parámetros  $\theta$  para minimizar una función de pérdida que mide las diferencias entre los datos observados y los pronosticados.
- El ajuste de los parámetros se realiza mediante el descenso de gradiente  $\nabla$  de la función de pérdida y  $\eta$  es la tasa de aprendizaje

$$f(\mathbf{x}; \theta) = f_L(f_{L-1}(\dots f_2(f_1(\mathbf{x}; \theta_1); \theta_2) \dots); \theta_L)$$

$$f_l(\mathbf{z}; \theta_l) = \sigma_l(\mathbf{W}_l \mathbf{z} + \mathbf{b}_l)$$

$$\theta = \{\mathbf{W}_l, \mathbf{b}_l\}_{l=1}^L$$

$$\mathcal{L}(f(\mathbf{x}; \theta), \mathbf{y})$$

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_{\theta} \mathcal{L}(f(\mathbf{x}; \theta), \mathbf{y})$$

$$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial E}{\partial w_{ij}}$$

# Redes neuronales artificiales

Dada una función con un vector de entrada  $\mathbf{x}$  de dimensión  $k$ , y  $\theta$  parámetros del modelo, que incluye los pesos sinápticos y sesgos, una **red neuronal artificial** (RNA) es un modelo computacional parametrizado basado en una sucesión de funciones compuestas:

- **Tasa de aprendizaje  $\eta$  alta:** puede llevar a que el algoritmo se salte el mínimo de la función de pérdida y no converja (demasiada exploración)
- **Tasa de aprendizaje  $\eta$  baja:** El modelo avanzará muy lentamente hacia el mínimo, lo que puede hacer que el proceso de entrenamiento sea extremadamente lento y puede quedar atrapado en un mínimo local o en una región plana del espacio de la función de pérdida

$$f(\mathbf{x}; \theta) = f_L(f_{L-1}(\dots f_2(f_1(\mathbf{x}; \theta_1); \theta_2) \dots); \theta_L)$$

$$f_l(\mathbf{z}; \theta_l) = \sigma_l(\mathbf{W}_l \mathbf{z} + \mathbf{b}_l)$$

$$\theta = \{\mathbf{W}_l, \mathbf{b}_l\}_{l=1}^L$$

$$\mathcal{L}(f(\mathbf{x}; \theta), \mathbf{y})$$

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_{\theta} \mathcal{L}(f(\mathbf{x}; \theta), \mathbf{y})$$

$$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial E}{\partial w_{ij}}$$

# Redes neuronales artificiales

Dada una función con un vector de entrada  $\mathbf{x}$  de dimensión  $k$ , y  $\theta$  parámetros del modelo, que incluye los pesos sinápticos y sesgos, una **red neuronal artificial** (RNA) es un modelo computacional parametrizado basado en una sucesión de funciones compuestas:

Una red neuronal típica tiene tres tipos de capas:

$$f(\mathbf{x}; \theta) = f_L(f_{L-1}(\dots f_2(f_1(\mathbf{x}; \theta_1); \theta_2) \dots); \theta_L)$$

1. **Capa de entrada (Input Layer):** Recibe los datos iniciales.
2. **Capas ocultas (Hidden Layers):** Procesan la información a través de combinaciones lineales y funciones no lineales.
3. **Capa de salida (Output Layer):** Produce el resultado final.

$$f_l(\mathbf{z}; \theta_l) = \sigma_l(\mathbf{W}_l \mathbf{z} + \mathbf{b}_l)$$

$$\theta = \{\mathbf{W}_l, \mathbf{b}_l\}_{l=1}^L$$

$$\mathcal{L}(f(\mathbf{x}; \theta), \mathbf{y})$$

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_{\theta} \mathcal{L}(f(\mathbf{x}; \theta), \mathbf{y})$$

$$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial E}{\partial w_{ij}}$$



# Redes neuronales artificiales profundas

Dada una función con un vector de entrada  $\mathbf{x}$  de dimensión  $k$ , y  $\theta$  parámetros del modelo, que incluye los pesos sinápticos y sesgos, una **red neuronal artificial** profunda es un modelo computacional parametrizado basado en una sucesión de funciones compuestas igual al de RNA, excepto que

- $L$  tiene un valor muy grande, específicamente en términos de capas ocultas.
- La optimización se realiza con técnicas de lotes (batch), el uso de funciones activación ReLU, regularización, y variantes modernas de descenso de gradiente estocástico como ADAM

$$f(\mathbf{x}; \theta) = f_L(f_{L-1}(\dots f_2(f_1(\mathbf{x}; \theta_1); \theta_2) \dots); \theta_L)$$

$$f_l(\mathbf{z}; \theta_l) = \sigma_l(\mathbf{W}_l \mathbf{z} + \mathbf{b}_l)$$

$$\theta = \{\mathbf{W}_l, \mathbf{b}_l\}_{l=1}^L$$

$$\mathcal{L}(f(\mathbf{x}; \theta), \mathbf{y})$$

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_{\theta} \mathcal{L}(f(\mathbf{x}; \theta), \mathbf{y})$$

$$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial E}{\partial w_{ij}}$$

# Funciones de activación no lineales

1. Función sigmoide. Mapea de 0 a 1. Tiene problemas de desvanecimiento de gradiente
2. Función tangente hiperbólica Mapea de -1 a 1. Tiene problemas de desvanecimiento de gradiente
3. ReLu (Rectified Linear Unit): Transforma todos los valores negativos en cero, manteniendo los valores positivos del gradiente. Resuelve el problema del desvanecimiento de gradiente pero puede generar neuronas muertas
4. Softmax: útil para targets multiclase, computacionalmente costosa
5. Swish: funciona mejor en redes profundas. Computacionalmente más costosa que ReLu

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

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

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

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}$$

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

# Funciones de activación no lineales

1. ELU: Evita el problema de la muerte de neuronas
2. SELU (Scaled Exponential Linear Unit): mantiene la normalización de las activaciones, normalmente  $\lambda \approx 1.0507$  y  $\alpha \approx 1.6733$
3. GELU (Gaussian Error Linear Unit): Util en NLP, particularmente en arquitecturas de transformers
4. Leaky ReLU: agrega un hiperparámetro  $\alpha$  para permitir el paso de valores negativos y evitar la muerte de neuronas. PReLU estima el hiperparámetro  $\alpha$  durante el entrenamiento

$$\text{ELU}(x) = \begin{cases} x & \text{si } x > 0 \\ \alpha(\exp(x) - 1) & \text{si } x \leq 0 \end{cases}$$

$$\text{SELU}(x) = \begin{cases} \lambda x & \text{si } x > 0 \\ \lambda \alpha(\exp(x) - 1) & \text{si } x \leq 0 \end{cases}$$

$$\begin{aligned} \text{GELU}(x) &= x \Phi(x) \\ &= x \frac{1}{2} \left[ 1 + \text{erf} \left( \frac{x}{\sqrt{2}} \right) \right] \\ \text{erf}(x) &= \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \end{aligned}$$

$$\text{Leaky ReLU}(x) = \begin{cases} x & \text{si } x > 0 \\ \alpha x & \text{si } x \leq 0 \end{cases}$$

# Capa densa (dense)

- Una capa densa es un capa completamente conectada (fully connected layer) en la que cada neurona está conectada a todas las neuronas de la capa anterior.
- La operación principal que realiza una capa densa es una multiplicación matricial seguida de una suma con un sesgo (bias) y, opcionalmente, la aplicación de una función de activación
- Se utiliza normalmente como capa final en una red profunda

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$$\mathbf{a} = f(\mathbf{z}) = f(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$z_i = \sum_{j=1}^k W_{ij} x_j + b_i$$

$$a_i = f(z_i) = f\left(\sum_{j=1}^k W_{ij} x_j + b_i\right)$$

$$a_i = f(z_i) = \frac{1}{1 + e^{-z_i}}$$

$$\mathbf{x} \in \mathbb{R}^k$$

$$\mathbf{W} \in \mathbb{R}^{n \times k}$$

$$\mathbf{b} \in \mathbb{R}^n$$

$$\mathbf{z} \in \mathbb{R}^n$$

$$\mathbf{a} \in \mathbb{R}^n$$

# Capa no densas

Las capa no densas no estan completamente conectadas. Dependiendo de la aplicación, pueden ser convolucionales, recurrentes o locales:

- En una capa convolucional se utiliza un filtro de dimensiones  $m \times n$  para detectar características relevantes

$$\mathbf{z}(i, j) = \sum_{m=1}^M \sum_{n=1}^N \mathbf{W}(m, n) \mathbf{x}(i + m - 1, j + n - 1) + \mathbf{b}$$

- En una capa recurrente hay una secuencia explícita (una dependencia temporal) entre capas

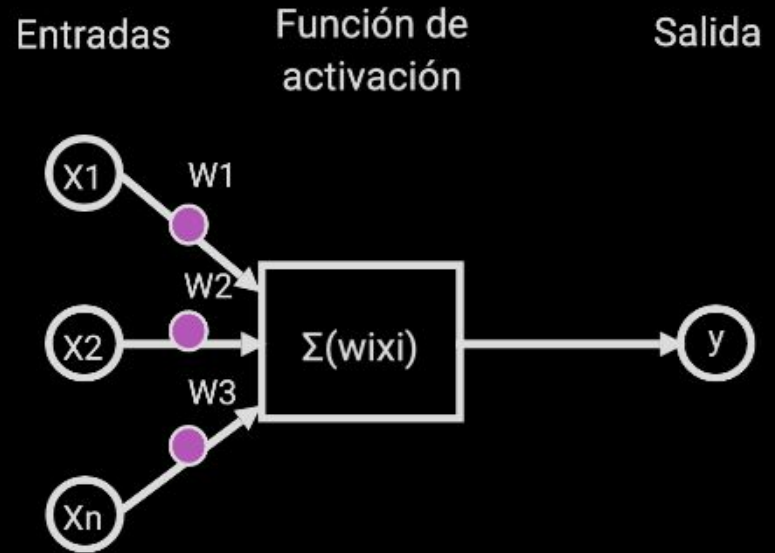
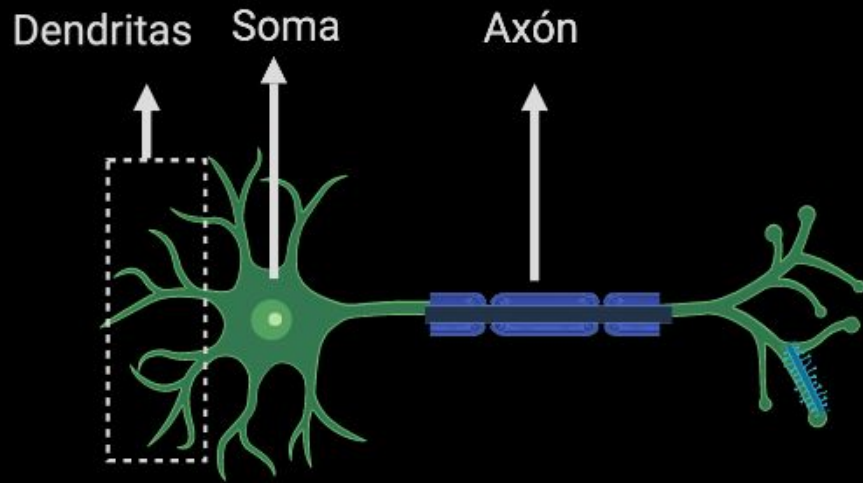
$$\mathbf{h}_t = f(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t + \mathbf{b})$$

$$\mathbf{z}(i, j) = \max_{m, n \in \text{Filtro}} \mathbf{x}(i + m, j + n)$$

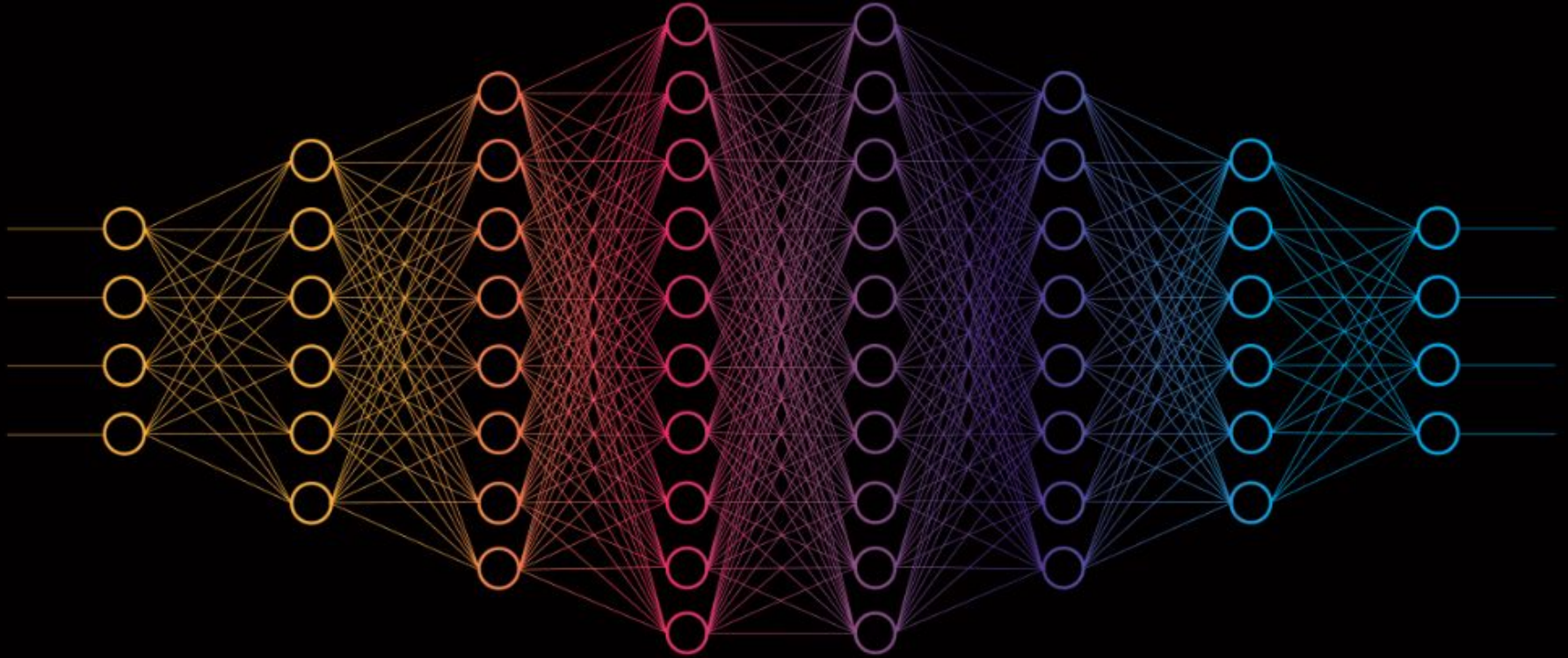
- Capas de pooling toman el valor máximo (Max Pooling) o promedio (Average Pooling) para reducir la dimensionalidad de las entradas

$$\mathbf{z}(i, j) = \frac{1}{M \times N} \sum_{m=1}^M \sum_{n=1}^N \mathbf{x}(i + m, j + n)$$

# Redes neuronales biológicas y redes neuronales artificiales



# Redes neuronales biológicas y redes neuronales artificiales



# Primero desarrollos de redes neuronales artificiales

- **1943: Neurona de McCulloch-Pitts.** Modelo matemático simple de una neurona artificial basada en una función escalón (función de activación no lineal, entradas  $x$  binarias, pesos  $w$ , y un umbral  $\theta$ )
- **1958: Perceptrón de Frank Rosenblatt.** Red neuronal de una sola capa capaz de aprender a clasificar patrones linealmente separables. Es una generalización del modelo de McCulloch-Pitts, donde se actualizan los pesos basándose en los errores cometidos durante el entrenamiento dadas una etiquetas  $d$  y una tasa de aprendizaje  $\eta$

$$y = \begin{cases} 1 & \text{si } \sum_{i=1}^n w_i x_i \geq \theta \\ 0 & \text{si } \sum_{i=1}^n w_i x_i < \theta \end{cases}$$

$$y = \begin{cases} 1 & \text{si } \sum_{i=1}^n w_i x_i + b > 0 \\ 0 & \text{si } \sum_{i=1}^n w_i x_i + b \leq 0 \end{cases}$$

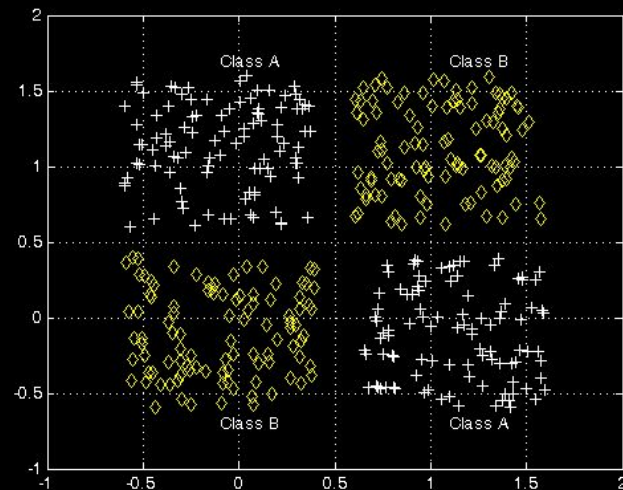
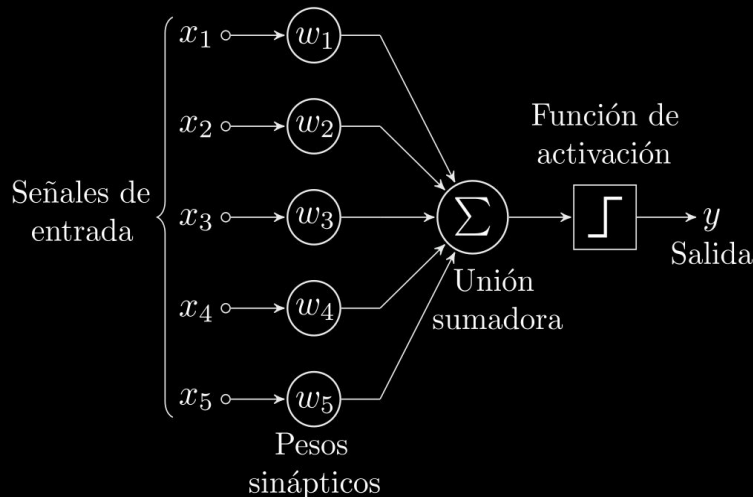
$$w_i \leftarrow w_i + \eta(d - y)x_i$$



# Limitaciones del perceptrón

**1969: Limitaciones del Perceptrón (Minsky y Papert).** En su libro "Perceptrons", Minsky y Papert demostraron que el perceptrón simple no puede resolver problemas no lineales, como la función XOR, porque el perceptrón es un modelo lineal y solo puede separar datos que son linealmente separables:

$$\text{XOR}(x_1, x_2) = \begin{cases} 1 & \text{si } (x_1 = 1 \text{ y } x_2 = 0) \text{ o } (x_1 = 0 \text{ y } x_2 = 1) \\ 0 & \text{si } x_1 = x_2 \end{cases}$$



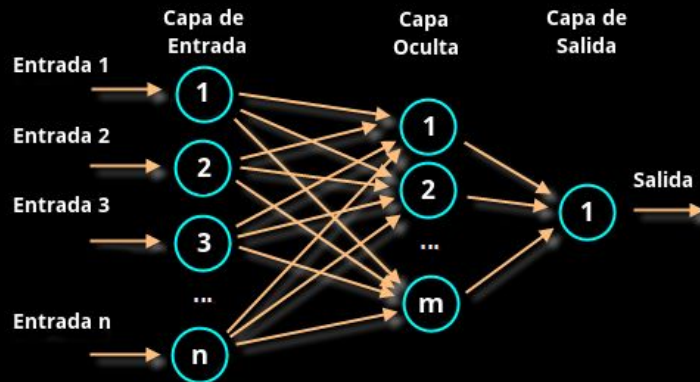
# Perceptrón multicapa

Extensión del perceptrón a redes neuronales de tres capas (entrada, oculta y de salida) que podía resolver problemas no-linealmente separables.

- La capa oculta está basada en **funciones de activación no-lineales**, como la sigmoideal o la tangente hiperbólica.
- Mas aún: el perceptrón multicapa puede aproximar arbitrariamente cualquier función continua, debido al **teorema de aproximación universal**
- Entrenar estas perceptrones multicapa resultaba extremadamente difícil debido a la falta de un método eficiente para ajustar los pesos en todas las capas.

$$\mathbf{z}_l = \mathbf{W}_l \mathbf{x}_{l-1} + \mathbf{b}_l$$
$$\mathbf{a}_l = \sigma_l(\mathbf{z}_l) \quad \sigma(z) = \frac{1}{1 + e^{-z}}$$


$$\mathbf{a}_L = \sigma_L(\mathbf{W}_L \mathbf{a}_{L-1} + \mathbf{b}_L)$$



# Intermission: teorema de aproximación universal


**PROMPT:** Dame la demostración matemática del teorema de aproximación universal, incluye referencias bibliográficas y la historia de quien lo propuso o quienes lo propusieron


Gemini Advanced ▾



Dame la demostración matemática del teorema de aproximación universal, incluye referencias bibliográficas y la historia de quien lo propuso o quienes lo propusieron

Mostrar versiones







**Demostración Matemática del Teorema de Aproximación Universal**

El Teorema de Aproximación Universal establece que una red neuronal artificial de alimentación hacia adelante (feedforward) con una sola capa oculta que contiene un número finito de neuronas puede aproximar cualquier función continua en un subconjunto compacto de  $\mathbb{R}^n$  con cualquier grado de precisión deseado, siempre que se le proporcione suficiente número de neuronas ocultas y se utilicen funciones

Introduce una petición aquí





# Intermission: teorema de aproximación universal

**PROMPT:** Dame la demostración matemática del teorema de aproximación universal, incluye referencias bibliográficas y la historia de quien lo propuso o quienes lo propusieron

Gemini **Advanced** ▾



## **Demostración (Esquema General):**

La demostración rigurosa del teorema de aproximación universal puede ser bastante técnica y requiere conocimientos avanzados de análisis funcional y teoría de la medida. Sin embargo, un esquema general de la demostración es el siguiente:

1. **Función de activación no lineal:** Se asume que la red neuronal utiliza una función de activación no lineal, como la sigmoide o la tangente hiperbólica. Esta no linealidad es crucial para la capacidad de la red de aproximar funciones complejas.
2. **Conjunto compacto:** Se considera un conjunto compacto  $K$  en  $\mathbb{R}^n$ . Un conjunto compacto es cerrado y acotado.

Introduce una petición aquí



# Intermission: teorema de aproximación universal

**PROMPT:** Dame la demostración matemática del teorema de aproximación universal, incluye referencias bibliográficas y la historia de quien lo propuso o quienes lo propusieron

Gemini Advanced ▾

## Historia y Proponentes:

El teorema de aproximación universal tiene una rica historia y ha sido objeto de estudio por varios investigadores. Algunos de los contribuyentes clave incluyen:

- **George Cybenko (1989):** Cybenko proporcionó una de las primeras demostraciones rigurosas del teorema para funciones de activación sigmoideas.
- **Kurt Hornik (1991):** Hornik extendió el resultado a una clase más amplia de funciones de activación y proporcionó una demostración más general.
- **Otros contribuyentes:** Varios otros investigadores, incluyendo a Moshe Leshno, Vladimir Vapnik, y Halbert White, han hecho contribuciones significativas al desarrollo y comprensión del teorema.

# Intermission: teorema de aproximación universal

**PROMPT:** Dame la demostración matemática del teorema de aproximación universal, incluye referencias bibliográficas y la historia de quien lo propuso o quienes lo propusieron

ChatGPT 4o ▾



Dame la demostración matemática del teorema de aproximación universal, incluye referencias bibliográficas y la historia de quien lo propuso o quienes lo propusieron



El **teorema de aproximación universal** es un resultado fundamental en el campo de las redes neuronales artificiales que establece que una red neuronal de una sola capa con una función de activación no lineal adecuada puede aproximar cualquier función continua definida en un espacio compacto con cualquier grado de precisión, siempre y cuando se permita un número suficiente de neuronas en la capa oculta.

# Intermission: teorema de aproximación universal

**PROMPT:** Dame la demostración matemática del teorema de aproximación universal, incluye referencias bibliográficas y la historia de quien lo propuso o quienes lo propusieron

ChatGPT 4o ▾



cualquier función continua definida en un espacio compacto con cualquier grado de precisión, siempre y cuando se permita un número suficiente de neuronas en la capa oculta.

## Historia y Contexto

El teorema de aproximación universal fue propuesto inicialmente por **George Cybenko** en 1989. Cybenko demostró que las redes neuronales de una sola capa con la función de activación sigmoideal son capaces de aproximar cualquier función continua en un espacio compacto. Posteriormente, este resultado fue generalizado por **Kurt Hornik**, **Maxwell Stinchcombe** y **Halbert White** en 1989 y 1990, quienes demostraron que la capacidad de aproximación no depende de la función de activación específica, sino de que sea una función no lineal y continua.

# Teorema de aproximación universal

## Cybenko-Hornik-Stinchcombe

- Intuitivamente: Hay cierta analogía con el Teorema de Aproximación de Weierstrass
- Proporciona un respaldo teórico sólido para el uso de RNA: son capaces de aproximar cualquier función continua con la precisión deseada, siempre que tengan suficientes neuronas en la capa oculta y una función de activación no lineal adecuada.

$$\begin{aligned} & C([0, 1]^n) \\ & f \in C([0, 1]^n) \\ & F(x) = \sum_{i=1}^h \omega_i \sigma(w_i x + b_i) \\ & \sup_{x \in [0, 1]^n} |f(x) - F(x)| < \epsilon \\ & \epsilon > 0 \end{aligned}$$

Cybenko, G. (1989). "Approximation by superpositions of a sigmoidal function". *Mathematics of Control, Signals and Systems*, 2(4), 303–314.

Hornik, K., Stinchcombe, M., & White, H. (1989). "Multilayer feedforward networks are universal approximators". *Neural Networks*, 2(5), 359–366.

Hornik, K. (1991). "Approximation capabilities of multilayer feedforward networks". *Neural Networks*, 4(2), 251–257.



# Backpropagation (propagación hacia atrás)

**1986: Propagación hacia atrás (Backpropagation):** Geoffrey Hinton, David Rumelhart, y Ronald Williams popularizaron el algoritmo de propagación hacia atrás, que permite entrenar redes neuronales con múltiples capas:

- **Paso 1: propagación hacia adelante.** Se calcula la salida de la red para una entrada dada  $x$ , propagando las activaciones desde la capa de entrada hacia la capa de salida.
- **Paso 2: propagación hacia atrás.** Se calcula de la función de pérdida, el gradiente de la función de pérdida respecto a las activaciones de cada capa, desde la capa de salida, y se retropropaga el error  $\delta$  de cada capa retrocediendo hasta la capa de entrada. Los pesos y sesgos en la dirección opuesta al gradiente

Los pasos 1 y 2 se repiten durante varias épocas (epochs) de entrenamiento

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

$$\nabla_{\mathbf{w}} \mathcal{L} = \left[ \frac{\partial \mathcal{L}}{\partial w_1}, \frac{\partial \mathcal{L}}{\partial w_2}, \dots, \frac{\partial \mathcal{L}}{\partial w_n} \right]$$

$$\mathbf{a}_L = \sigma_L(\mathbf{W}_L \mathbf{a}_{L-1} + \mathbf{b}_L)$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_l} = \delta_l \mathbf{a}_{l-1}^T$$

$$\delta_l = (\mathbf{W}_{l+1}^T \delta_{l+1}) \odot \sigma'_l(\mathbf{z}_l)$$

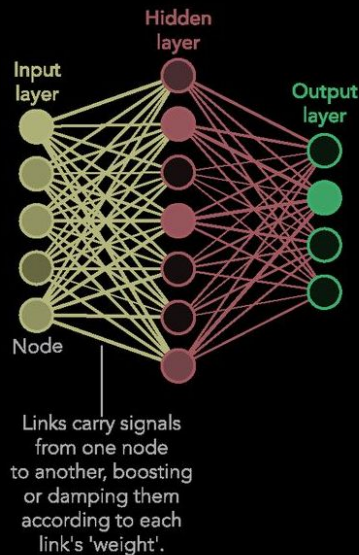
$$\mathbf{W}_l \leftarrow \mathbf{W}_l - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}_l}$$

$$\mathbf{b}_l \leftarrow \mathbf{b}_l - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{b}_l}$$

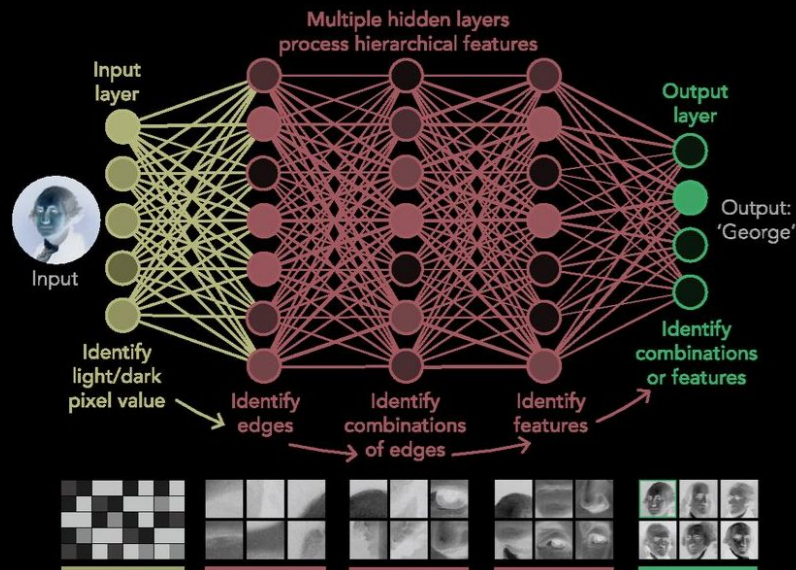
# Redes neuronales profundas

**Década de 2010:** Deep Learning: Debido al aumento del poder computacional y el acceso a grandes volúmenes de datos (big data), las redes neuronales profundas (Deep Learning) comenzaron a dominar muchas áreas de la inteligencia artificial, como la visión por computadora:

1980S-ERA NEURAL NETWORK



DEEP LEARNING NEURAL NETWORK



# Adaptive moment estimation (ADAM)

Algoritmo de optimización muy utilizado en el entrenamiento de modelos de aprendizaje profundo.

Adapta las tasas de aprendizaje para cada parámetro utilizando estimaciones del primer y segundo momento de los gradientes.

Hiperparámetros:

- Tasa de aprendizaje:  $\alpha$
- Parámetros de decaimiento:  $\beta_1, \beta_2$
- Término de estabilidad numérica:  $\epsilon$
- Contador de iteraciones:  $t = 1, 2, \dots, T$

$$\theta_0 \quad m_0 = 0, v_0 = 0 \quad \alpha, \beta_1, \beta_2, \epsilon$$

$$g_t = \nabla_{\theta} L(\theta_{t-1})$$

$$L(\theta) = \frac{1}{n} \sum_{i=1}^n \left( h_{\theta}(x^{(i)}) - y^{(i)} \right)^2$$

$$L(\theta) = -\frac{1}{n} \sum_{i=1}^n \left[ y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right]$$

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

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

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

# Redes Neuronales Recurrentes

- Las redes neuronales recurrentes (RNNs) tienen una estructura matemática que les permite procesar secuencias de datos como las de series de tiempo.
- La conexión recurrente de las RNN implica que la salida de una capa en el tiempo  $t$  puede ser utilizada como entrada en el tiempo  $t+1$
- Se estiman con Backpropagation Through Time (BPTT), una extensión de BPTT, desde  $T$  hasta 1, en el que las capas ocultas en  $t$  se actualizan con las de  $t-1$ , lo que puede llevar al desvanecimiento de gradientes o a la explosión de gradientes

$$h_t = f(W_{hx}x_t + W_{hh}h_{t-1} + b_h)$$

$$y_t = g(W_{hy}h_t + b_y)$$

$$L = \sum_{t=1}^T L_t(y_t, \hat{y}_t)$$

$$\frac{\partial L}{\partial W_{hx}} = \sum_{t=1}^T \frac{\partial L}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial W_{hx}}$$

# Redes Neuronales Recurrentes: LSTM y GRU

Para superar la incapacidad de las RNN tradicionales de recordar información relevante en secuencias largas y mitigar los desvanecimiento de gradientes o a la explosión de gradientes, se desarrollaron variantes de RNNs: **LSTM (Long Short-Term Memory)** y **GRU (Gated Recurrent Unit)**, que incluyen mecanismos para controlar el flujo de información y permiten que la red recuerde dependencias a largo plazo más efectivamente:

- **LSTM:** están basadas en celdas de memoria e incluyen funciones (puerta) de olvido, de entrada, y de salida.
- **GRU:** Simplifican la estructura de las LSTM al combinar algunas de las puertas y eliminar la celda de memoria separada, solamente tienen una función (puerta) de actualización (que decide cuánto del estado oculto anterior debe ser retenido y cuánta nueva información debe ser añadida) una función de reinicio que controla cuánta de la información pasada influye en la nueva información que se genera) y un estado oculto (que une las funciones de salida y memoria)

# Redes Neuronales Recurrentes: LSTM

- En un RNA de tipo LSTM existe una celda de memoria dinámica que depende de una función de olvido y una función de entradas (inputs)
- Una función tangente hiperbólica se usa para calcular candidatas a la nueva celda de memoria
- Una función de salida (output) se aplica para decidir cuánta de la información almacenada en la celda de memoria se utiliza para generar la salida  $h$ , que se pasará como entrada a la siguiente unidad en la secuencia.

$$C_t = f_t C_{t-1} + i_t \tilde{C}_t$$

$$f_t = \sigma(W_f [h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(W_i [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C [h_{t-1}, x_t] + b_C)$$

$$o_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t \tanh(C_t)$$

# Redes Neuronales Recurrentes: GRU

- No tiene una celda de memoria separada
- La puerta de actualización (z) controla cuánto del estado oculto anterior se debe llevar al siguiente paso de tiempo.
- La puerta de reinicio decide cuánta información del estado oculto anterior se debe olvidar antes de calcular el nuevo estado.
- El estado intermedio es una versión "propuesta" del nuevo estado oculto  $h$ . Se calcula utilizando la entrada actual y el estado oculto anterior, modulados el reinicio

$$z_t = \sigma(W_z [h_{t-1}, x_t] + b_z)$$

$$r_t = \sigma(W_r [h_{t-1}, x_t] + b_r)$$

$$\tilde{h}_t = \tanh(W_h [r_t h_{t-1}, x_t] + b_h)$$

$$h_t = (1 - z_t)h_{t-1} + z_t\tilde{h}_t$$

El estado oculto  $h$  es una mezcla controlada de la información pasada y la nueva información, donde  $z$  decide cuánto del pasado se mantiene y cuánto de la nueva información se añade.

# Regularización en redes neuronales artificiales

- En redes neuronales artificiales y en aprendizaje profunda se aplica regularización por drop-out y regularizaciones L1 y L2
- Dropout es una técnica de regularización que "apaga" aleatoriamente un porcentaje de neuronas durante el entrenamiento para evitar el sobreajuste:
  - **Dropout alto:** Puede prevenir el sobreajuste, pero también puede hacer que el modelo subajuste si se aplica en exceso.
  - **Dropout bajo:** Permite que el modelo aprenda más detalles, pero aumenta el riesgo de sobreajuste.
  - Un valor común de Dropout está entre 0.2 y 0.5, dependiendo del problema y la cantidad de datos.



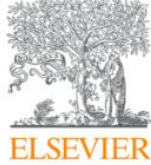
# Laboratorios

- **AIMLDL\_0501.ipynb**: Perceptron multicapa (redes neuronales artificiales multicapa con funciones de activación no-lineales)
- **AIMLDL\_0502.ipynb**: Perceptron multicapa (redes neuronales artificiales multicapa con funciones de activación no-lineales): experimento de pseudo-pronóstico fuera-de-muestra de criptomonedas.
- **AIMLDL\_0503.ipynb**: Red neuronal LSTM sin regularización para criptomonedas (Bitcoin y Ethereum)
- **AIMLDL\_0504.ipynb**: Redes neuronales LSTM y GRU no regularizadas
- **AIMLDL\_0505.ipynb**: Redes neuronales LSTM con regularización
- **AIMLDL\_0506.ipynb**: Redes neuronal LSTM profunda (deep learning)

# Ejemplos de aplicaciones de Deep Learning


Se comparan modelos/algoritmos de ML con RNA de aprendizaje profundo (deep learning) para el pre-screening de cancer de mama:

<https://www.sciencedirect.com/science/article/pii/S2352914823001636?via%3Dihub>






Informatics in Medicine Unlocked

Volume 41, 2023, 101317



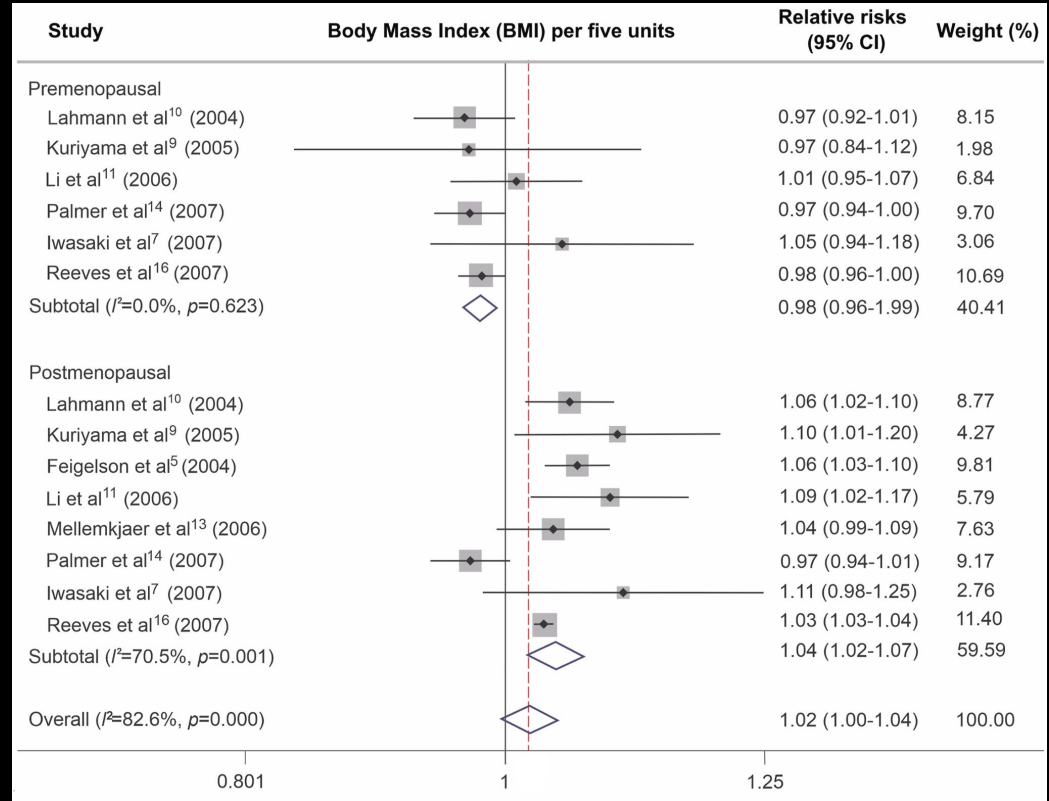
## Deep learning algorithms for the early detection of breast cancer: A comparative study with traditional machine learning

Rolando Gonzales Martinez <sup>a</sup>  , Daan-Max van Dongen <sup>b</sup>

Show more 

# Ejemplos de aplicaciones de Deep Learning

- Screening de cancer de mama: mamogramas, biopsias
- Pre-screening: datos demográficos, antropométricos, riesgos relativos (GBD) y obtenidos de muestras de sangre rutinarias



# Ejemplos de aplicaciones de Deep Learning

- Screening de cancer de mama: mamogramas, biopsias
- Pre-screening: datos demográficos, antropométricos, riesgos relativos (GBD) y obtenidos de muestras de sangre rutinarias

**Table 1**

Descriptive statistics of the database.

	mean	std.dev.	min	max
Age (years)	57.30	16.11	24	89
BMI	27.58	5.02	18.37	38.58
Glucose (mg/dL)	97.79	22.53	60.00	201.00
Insulin ( $\mu$ U/mL)	10.01	10.07	2.43	58.46
HOMA	2.69	3.64	0.47	25.05
Leptin (ng/mL)	26.62	19.18	4.31	90.28
Adiponectin ( $\mu$ g/mL)	10.18	6.84	1.66	38.04
Resistin (ng/mL)	14.73	12.39	3.21	82.10
RRs [19]	1.51	0.48	0.40	2.44
RRs GBD (center)	1.01	0.10	0.89	1.09
RRs GBD (lower)	0.97	0.08	0.87	1.04
RRs GBD (upper)	1.05	0.11	0.91	1.14
High BMI (binary)	0.34	0.47	0	1
Obesity (binary)	0.32	0.47	0	1

# Ejemplos de aplicaciones de Deep Learning

Selección de características (features):

- Feature selection se realizó con boosting de gradiente extremo.

**Table 2**

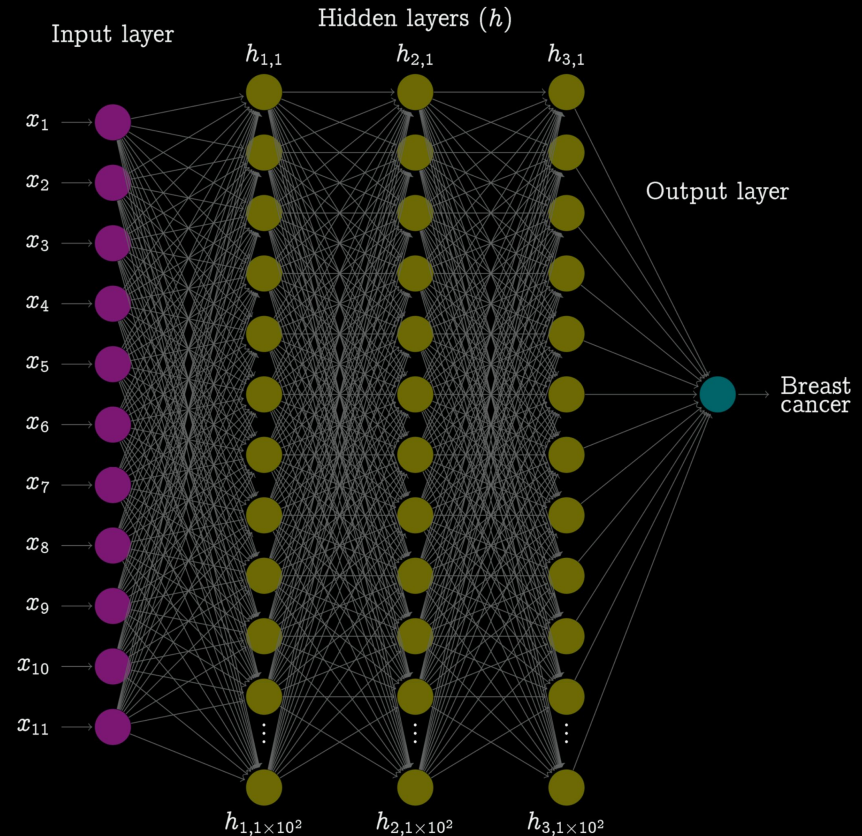
Recursive MRMR feature selection with SULOV-gradient boosting.

	description	frequency	min $\rho$	mean $\rho$	std.dev. $\rho$
$x_1$	Age (years)	97	0.00	0.49	0.28
$x_2$	Resistin (ng/mL)	90	0.00	0.51	0.28
$x_3$	RRs GBD (upper)	77	0.05	0.55	0.27
$x_4$	Glucose (mg/dL)	76	0.21	0.59	0.22
$x_5$	Adiponectin ( $\mu$ g/mL)	75	0.23	0.60	0.22
$x_6$	High BMI (binary)	65	0.27	0.64	0.20
$x_7$	MCP-1 (pg/dL)	65	0.26	0.65	0.19
$x_8$	Leptin (ng/mL)	64	0.30	0.65	0.19
$x_9$	RRs [19]	59	0.37	0.68	0.17
$x_{10}$	Obesity (binary)	59	0.11	0.55	0.28
$x_{11}$	Insulin ( $\mu$ U/mL)	33	0.05	0.53	0.26
	BMI	30	0.62	0.81	0.11
	HOMA	29	0.69	0.83	0.09
	RRs GBD (center)	8	0.86	0.90	0.04
	RRs GBD (lower)	8	0.21	0.81	0.25

# Ejemplos de aplicaciones de Deep Learning

En la arquitectura de la red neuronal profunda:

- Funciones de activación lineal rectificadas (ReLU)
- 3 capas ocultas
- 102 nodos en cada capa oculta, con regularizadores L1 y L2 para evitar el sobreajuste

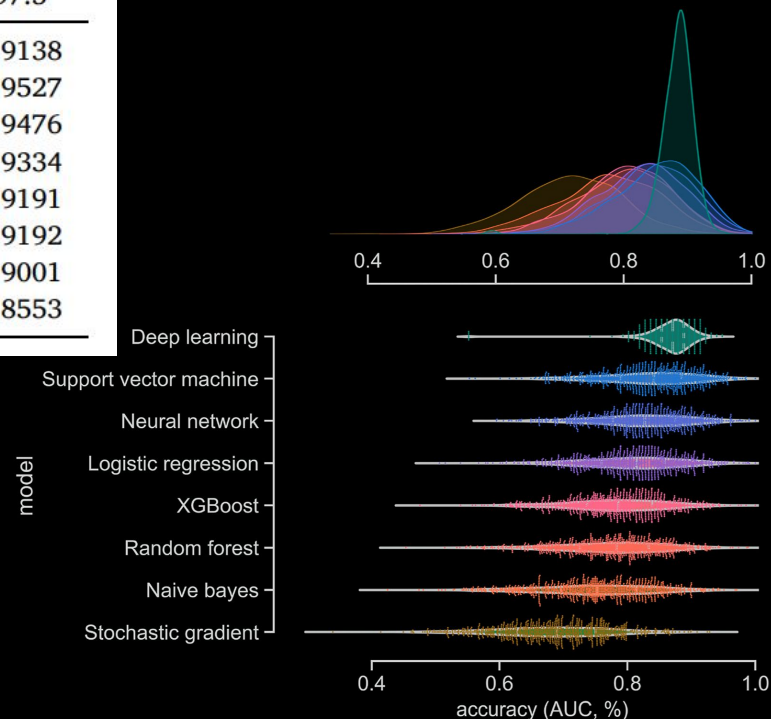


# Ejemplos de aplicaciones de Deep Learning

**Table 3**

AUC of the machine learning and deep learning algorithms.

Model	mean	std.dev.	p2.5	p97.5
Deep learning	0.8699	0.0345	0.8190	0.9138
Support vector machines	0.8344	0.0711	0.6768	0.9527
Neural network	0.8232	0.0720	0.6667	0.9476
Logistic regression	0.8078	0.0741	0.6569	0.9334
XGBoost	0.7834	0.0755	0.6263	0.9191
Random forest	0.7763	0.0804	0.6035	0.9192
Naive bayes	0.7504	0.0869	0.5686	0.9001
Stochastic gradient	0.6861	0.0860	0.5120	0.8553





# Ejemplos de aplicaciones de Deep Learning

**Table 4**

Performance metrics based on the confusion matrix.

Model	Derivations from confusion matrix*					
	TPR	TNR	FNR	FPR	FDR	FOR
Deep learning	0.9220 (0.0597)	0.8021 (0.1118)	0.0780 (0.0597)	0.1979 (0.1118)	0.1425 (0.0522)	0.0983 (0.0655)
SVM	0.7939 (0.1183)	0.6747 (0.1381)	0.2061 (0.1183)	0.3253 (0.1381)	0.2501 (0.1065)	0.2672 (0.1399)
Neural network	0.7638 (0.1148)	0.7135 (0.1324)	0.2362 (0.1148)	0.2865 (0.1324)	0.2331 (0.1051)	0.2851 (0.1284)
Logistic regression	0.7329 (0.1243)	0.7311 (0.1292)	0.2671 (0.1243)	0.2689 (0.1292)	0.2291 (0.108)	0.3055 (0.1319)
XGBoost	0.7537 (0.1158)	0.6723 (0.1342)	0.2463 (0.1158)	0.3277 (0.1342)	0.2604 (0.1029)	0.3061 (0.1327)
Random forest	0.6892 (0.1267)	0.7038 (0.1365)	0.3108 (0.1267)	0.2962 (0.1365)	0.2575 (0.1144)	0.3477 (0.1263)
Naive bayes	0.5313 (0.1296)	0.8023 (0.1131)	0.4687 (0.1296)	0.1977 (0.1131)	0.2303 (0.126)	0.4147 (0.1083)
Stochastic gradient	0.7126 (0.1394)	0.6518 (0.1587)	0.2874 (0.1394)	0.3482 (0.1587)	0.2792 (0.1106)	0.3412 (0.1409)

(\*) TPR: True positive rate, TNR: true negative rate, FNR: false negative rate, FPR: false positive rate, FDR: false detection rate, FOR: false omission rate. Rates are averages of the  $1 \times 10^3$  k-fold cross validations. Standard deviations in brackets below each average rate. SVM: Support Vector Machines.