

Cursos Extraordinarios

verano 2025

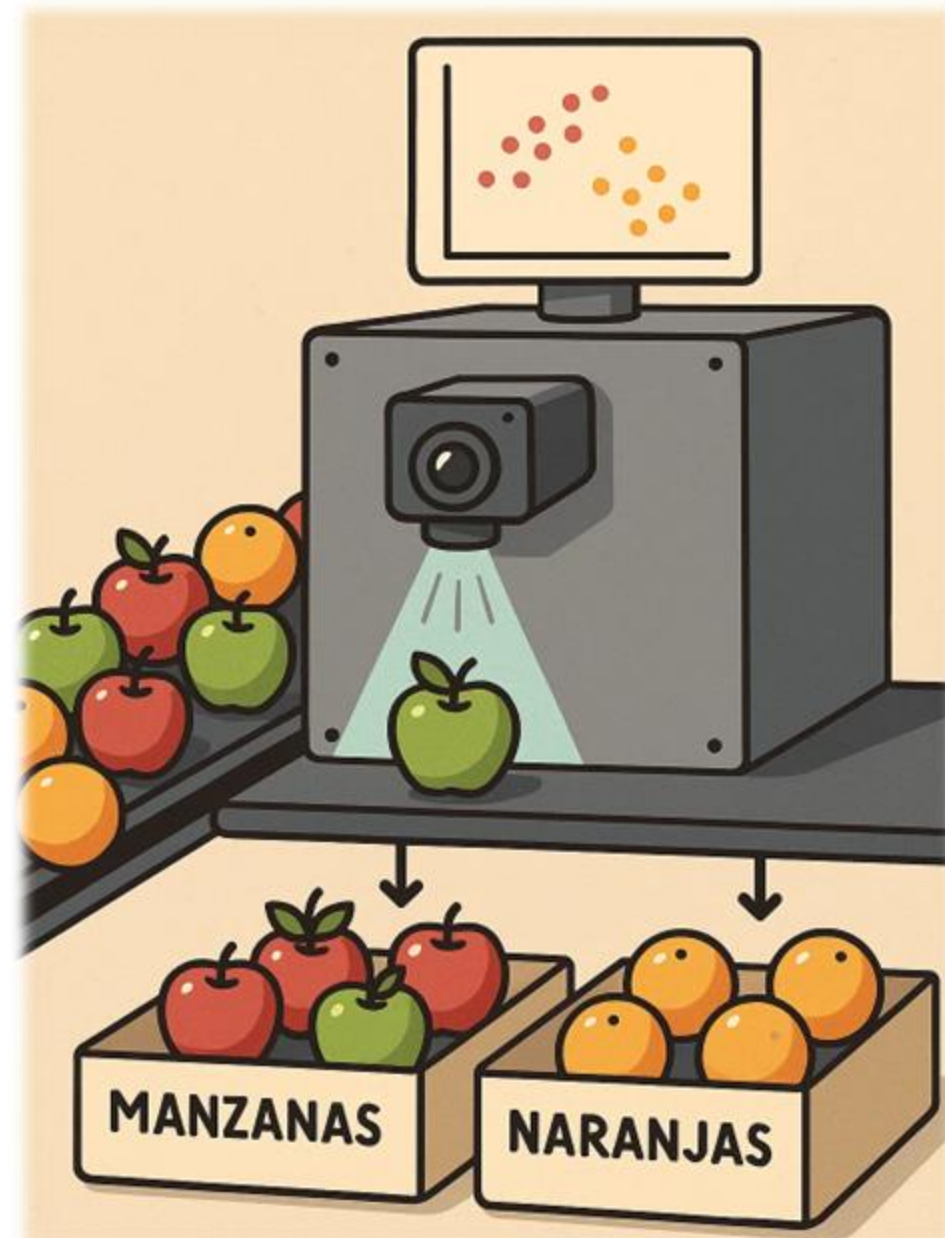
**“Inteligencia Artificial y Grandes
Modelos de Lenguaje: Funcionamiento,
Componentes Clave y Aplicaciones”**

Zaragoza, del 30 de junio al 02 de julio de 2025

Introducción al Aprendizaje Automático y las Redes Neuronales

Aprendizaje Automático

- Desarrollo de **sistemas capaces** de realizar **tareas de forma automática** mediante el **análisis de datos**.
- No necesitan ser programados explícitamente para cada tarea.
 - Cada tarea precisa de sus propios datos
- **Deep Learning:**
 - Uso de Redes Neuronales Artificiales (generalmente de múltiples capas)



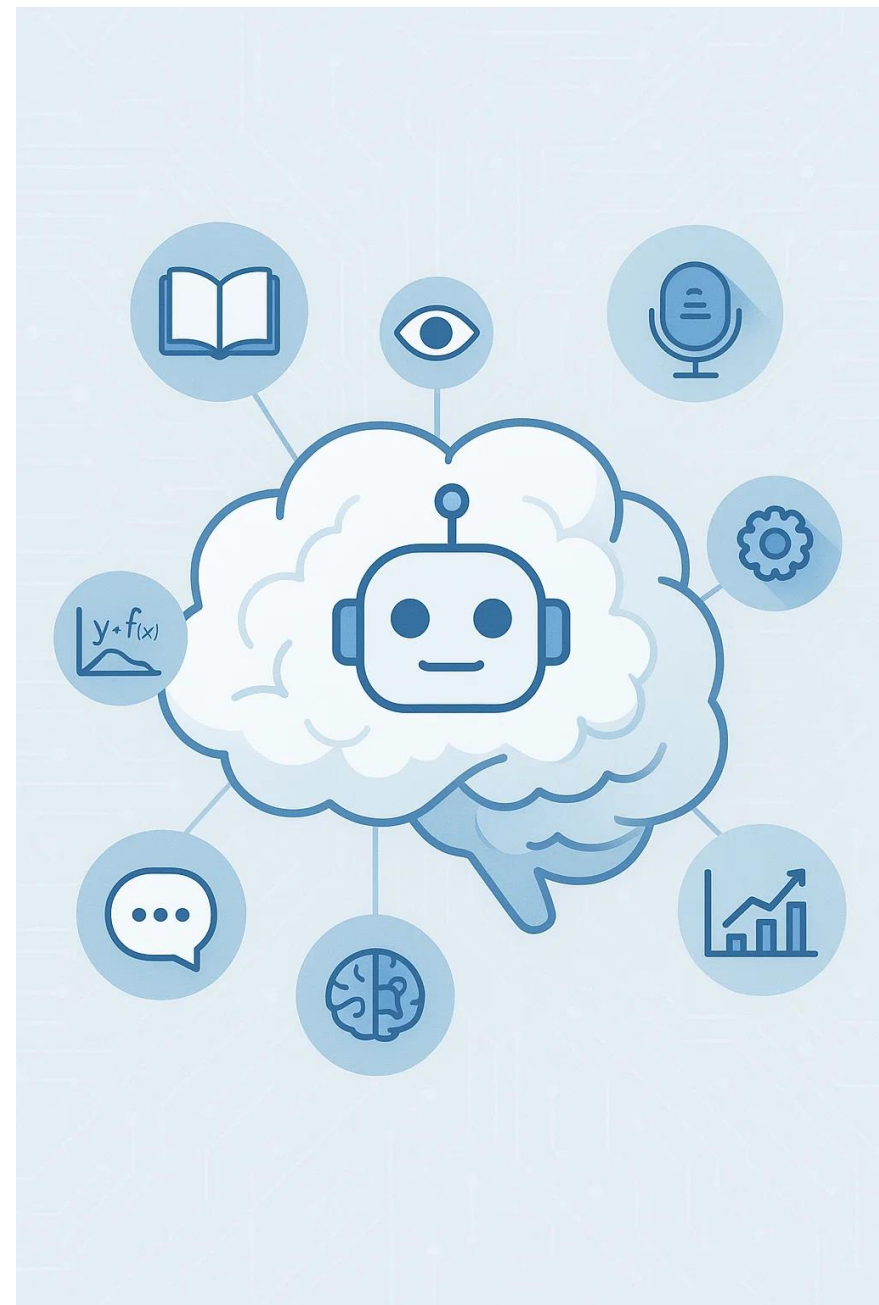
Aprendizaje Automático

- Asociado actualmente al concepto de Inteligencia Artificial

Capacidad de las máquinas para realizar tareas que normalmente requieren destrezas humanas.

Esto incluye:

**Aprendizaje,
percepción,
razonamiento,
reconocimiento automático del habla,
comprensión del lenguaje natural,
toma de decisiones, ...**



EARLY PIONEERS *of* ARTIFICIAL INTELLIGENCE

THE THEORY



McCulloch & Pitts
1943



Frank Rosenblatt
1957



Norbert Wiener
1948



Alan Turing
1950



Mark I,
1944

1 operation
every 3 seconds



ADALINE
(1960)

Widrow & Hoff
1960

THE MACHINES

Explosión Tecnológica

Capacidad Computacional

Almacenamiento Digital

Comunicaciones / Difusión

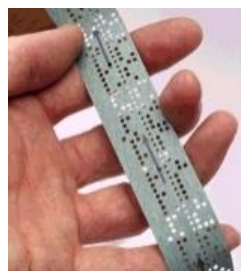


Explosión Tecnológica

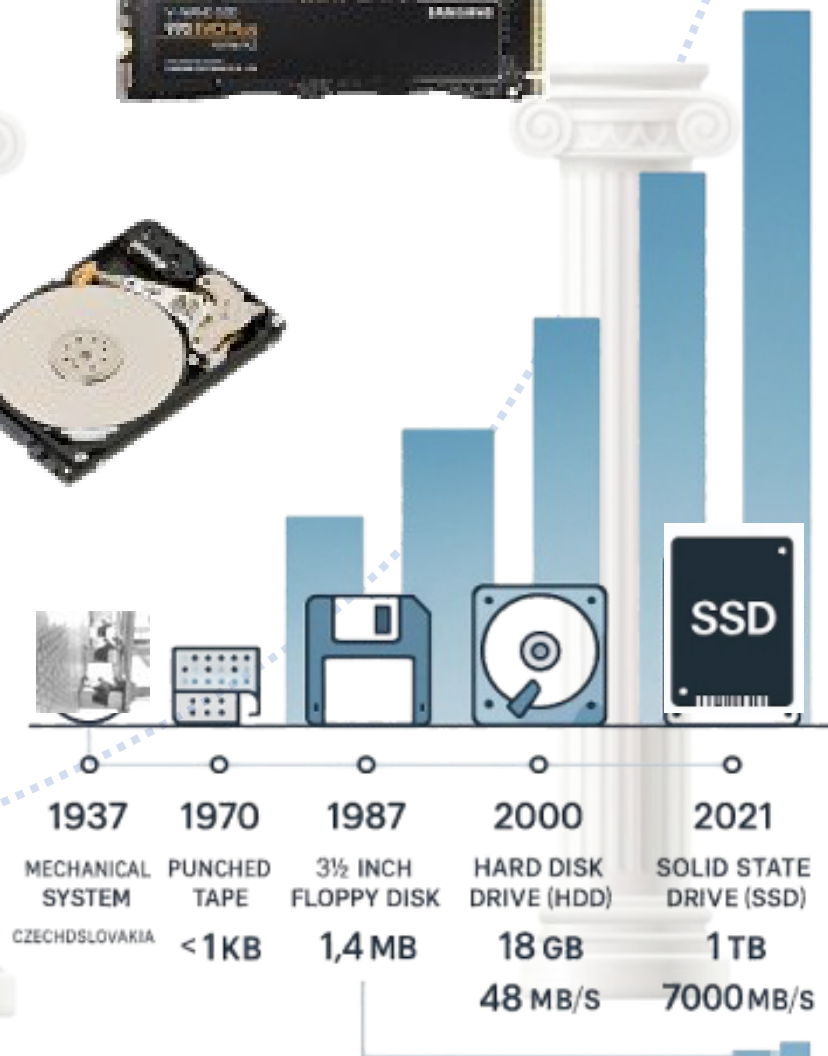


Explosión Tecnológica

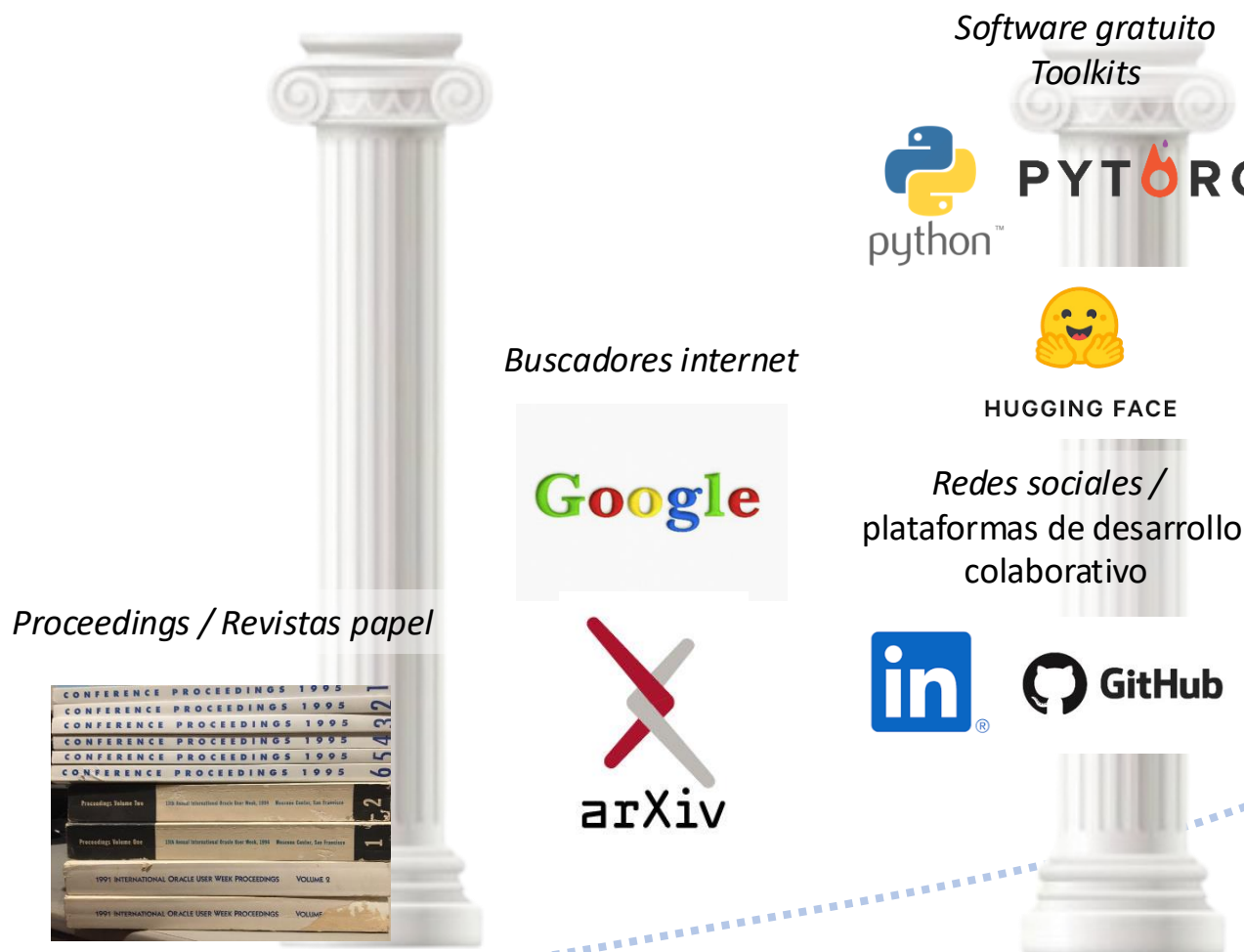
Sistema mecánico 1937
(República Checa)



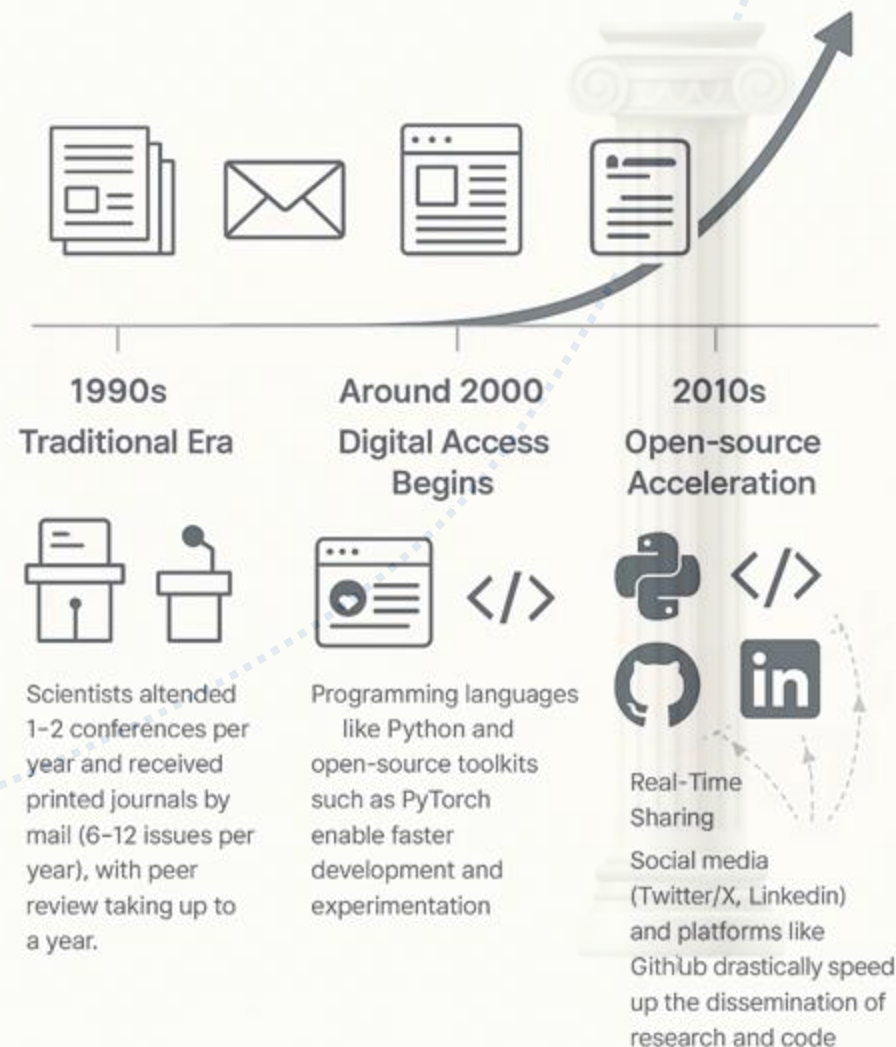
EVOLUTION OF DIGITAL STORAGE



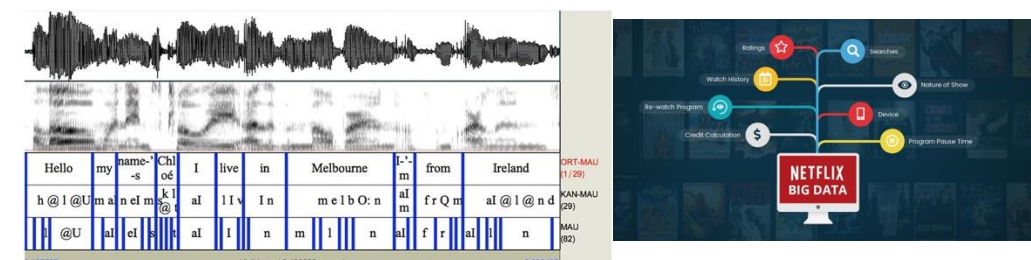
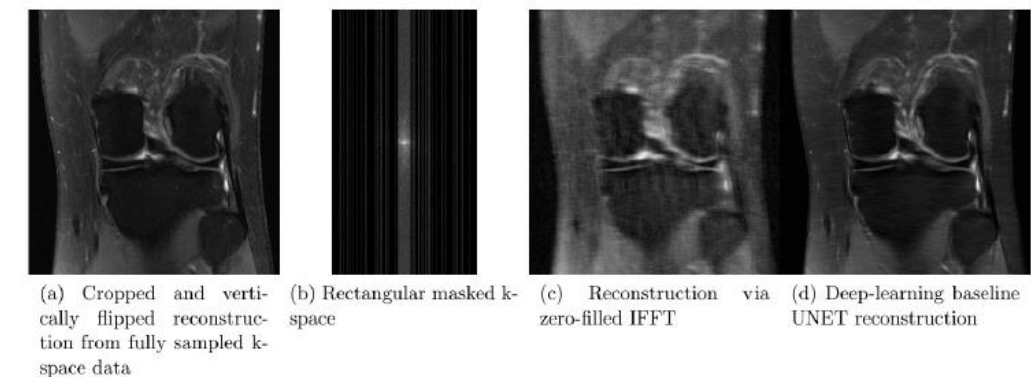
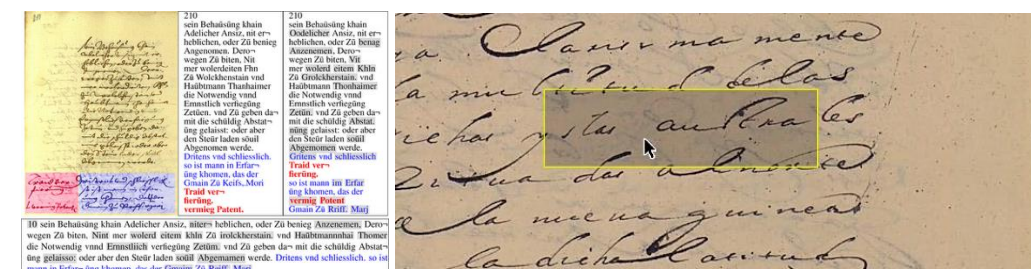
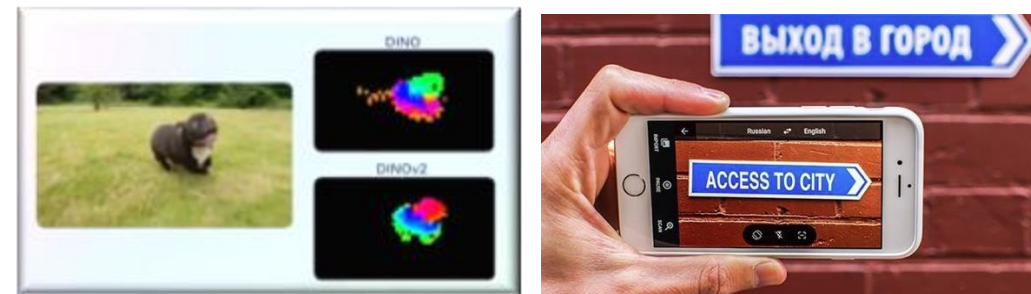
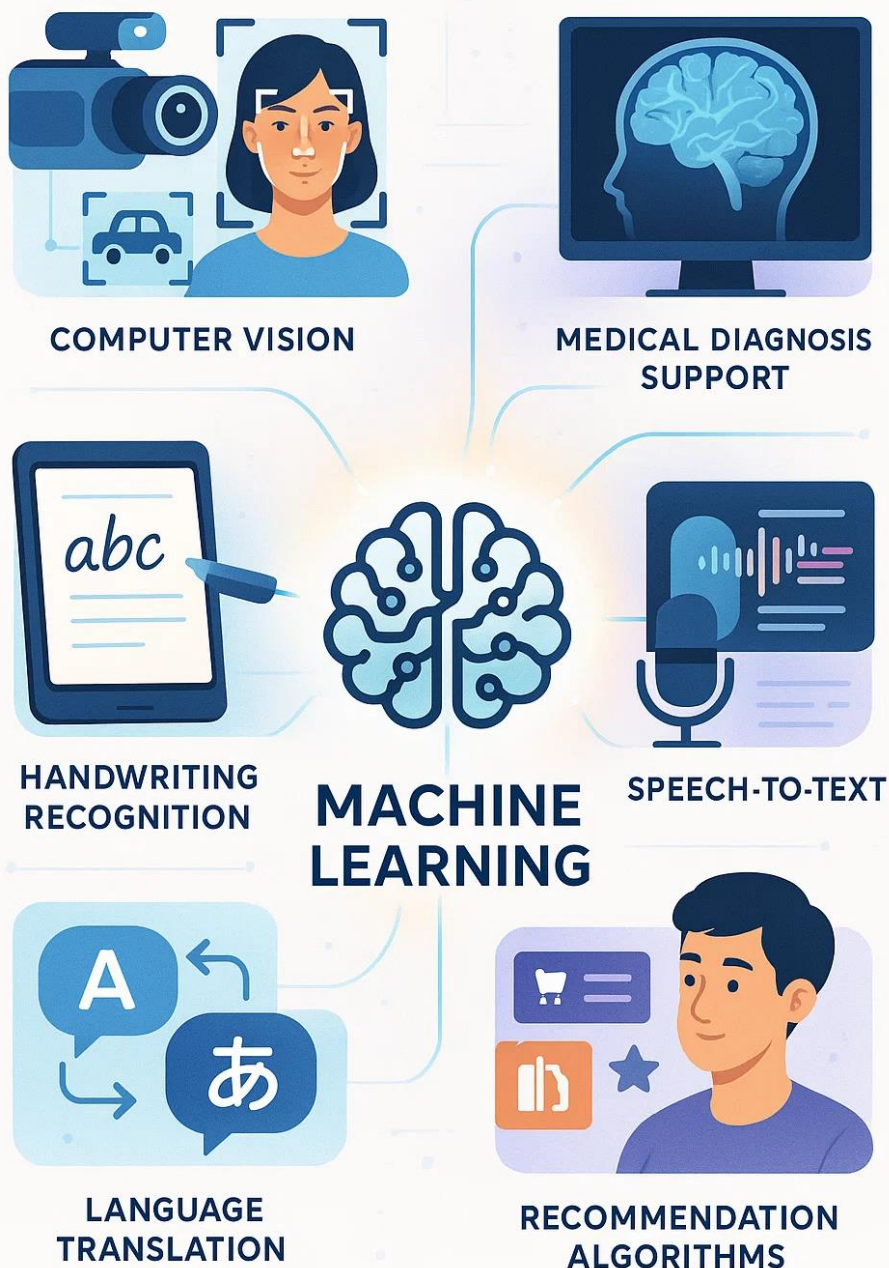
Explosión Tecnológica



The Evolution of Scientific and Technical Communication



Campos de Aplicación



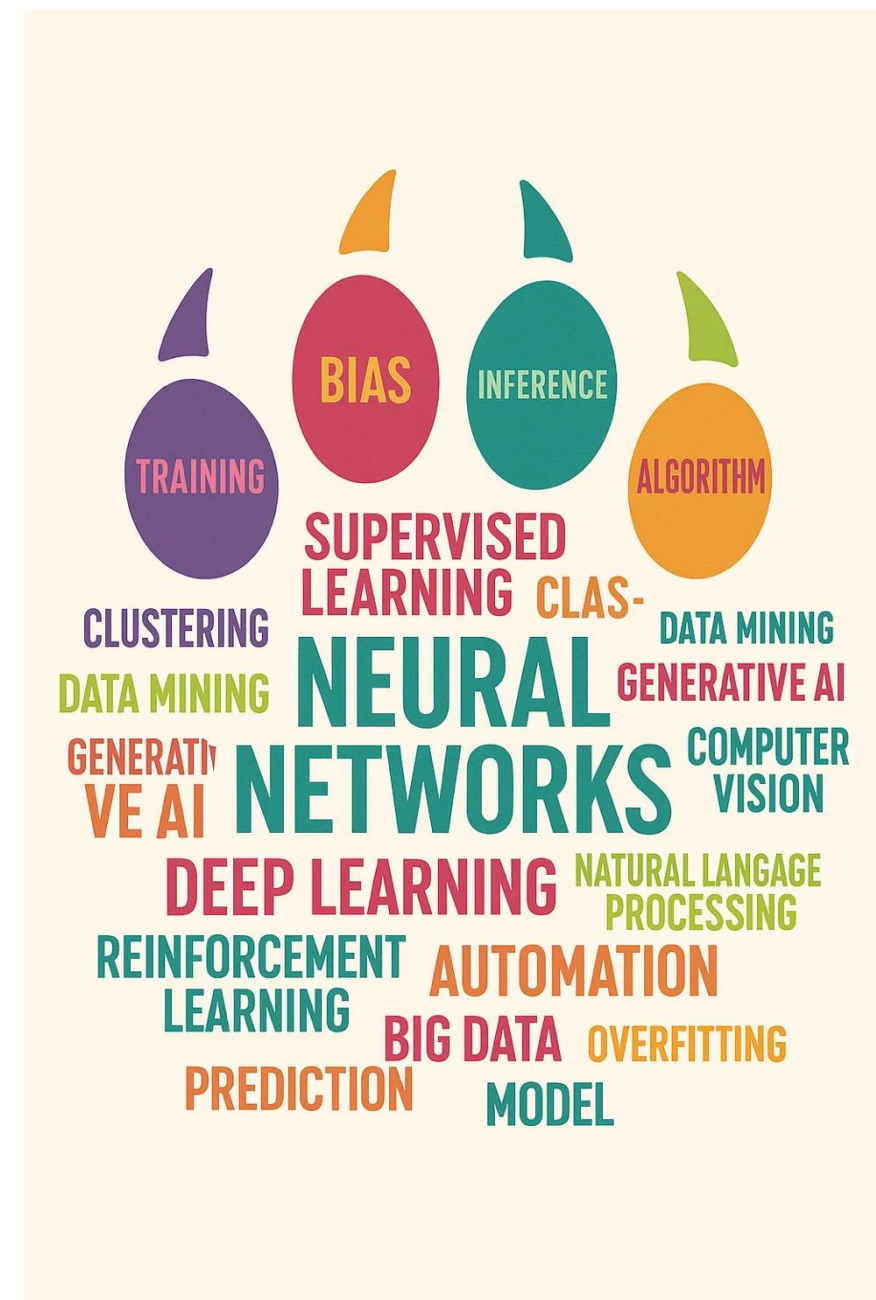
Aprendizaje Automático

- “*Learn from data*”
- Encontrar patrones y tendencias, entendiendo
“ lo que nos dicen los datos ”
- Sistema automático que aprende de la **Experiencia (E)** para realizar una **Tarea (T)** evaluada a través de una **Métrica (M)** establecida.

Tom M. Mitchell

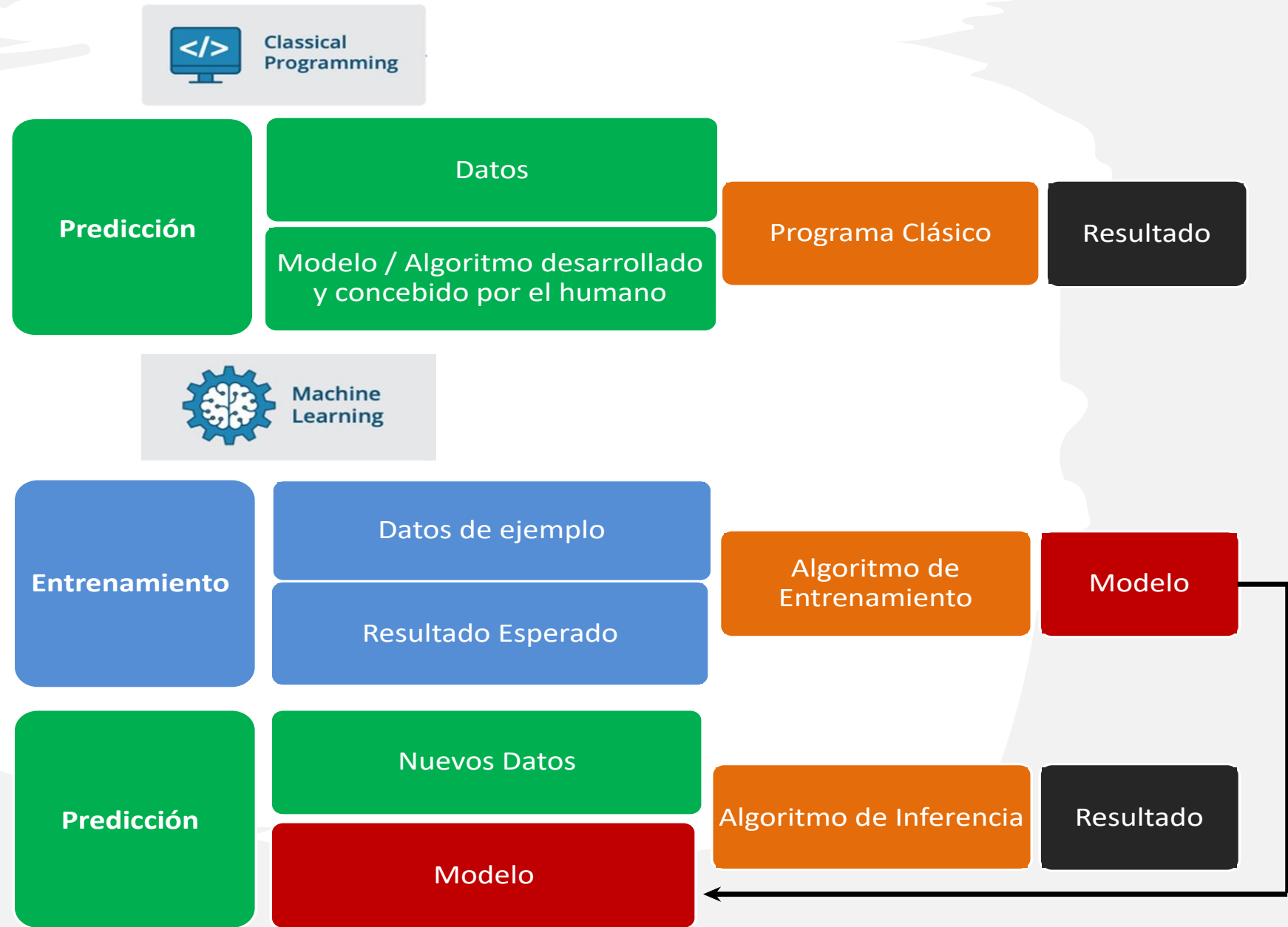
American computer scientist; Founders University Professor
[Carnegie Mellon University](#) (CMU).

Tom Mitchell [T. Mitchell. Machine Learning. McGraw Hill, 1997]



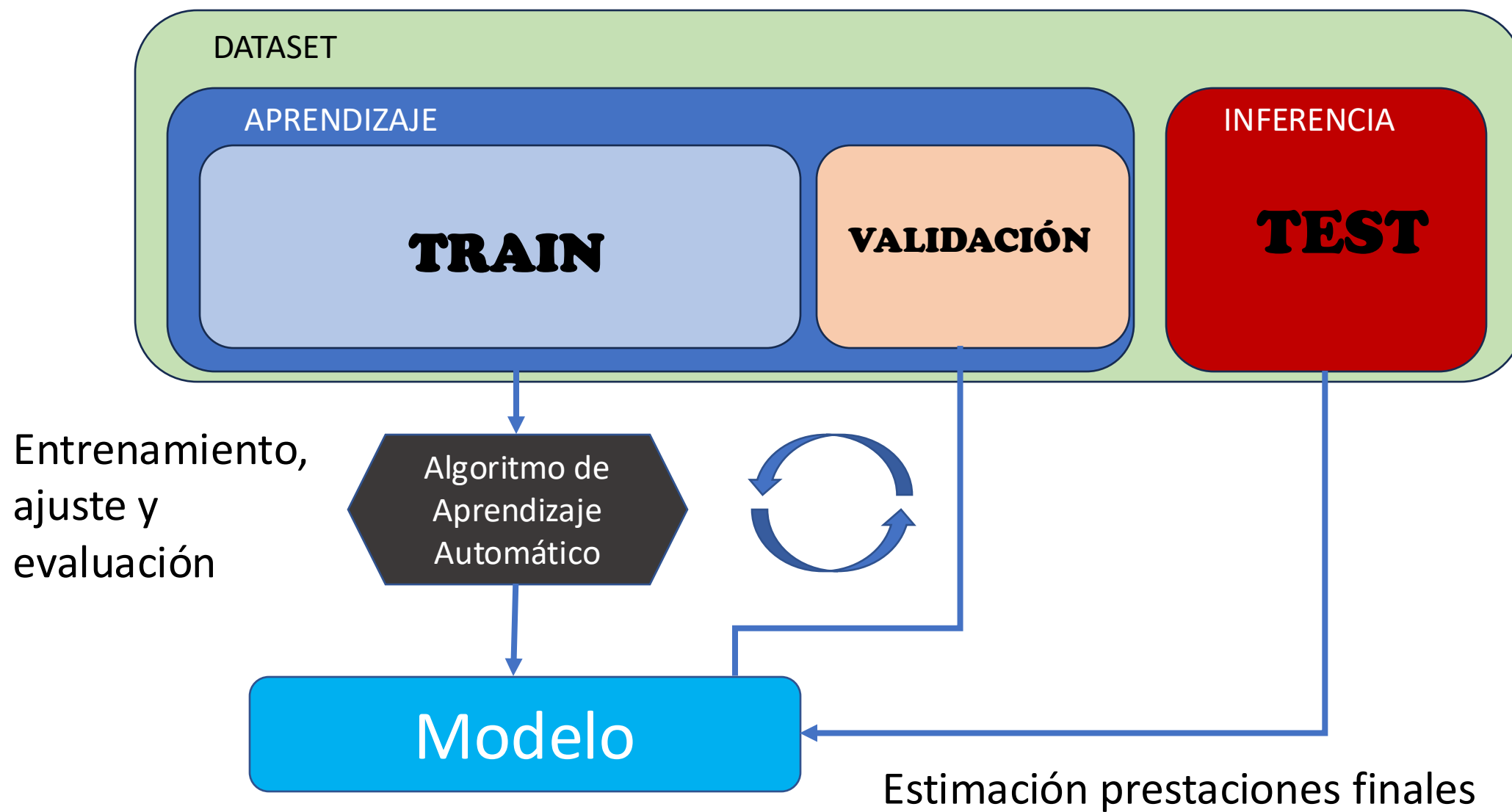
Aprendizaje Automático

Ofrece soluciones a **Problemas** demasiado **complejos** como para ser resueltos por **programas clásicos** concebidos por humanos



Diseño Experimental

- **Uso de los datos:**



Tipos de Tareas

- **Clasificación:**
 - Simple:
 - Múltiple:
- **Regresión:**
 - Simple:
 - Múltiple:

Tipos de Tareas

- **Clasificación:**

CLASIFICACIÓN SIMPLE

- Simple:

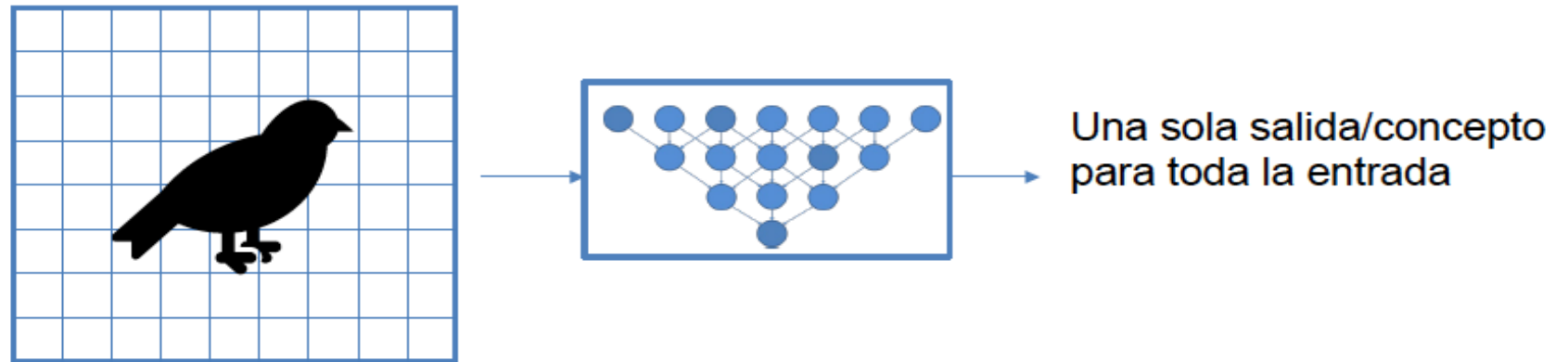
¿Qué **CONCEPTO** hay tras una imagen/texto/audio ?

- Múltiple:

- **Regresión:**

- Simple:

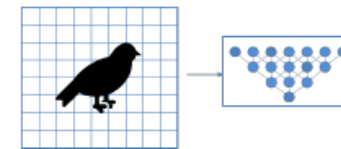
- Múltiple:



Tipos de Tareas

- Clasificación:**

CLASIFICACIÓN SIMPLE



- Simple:

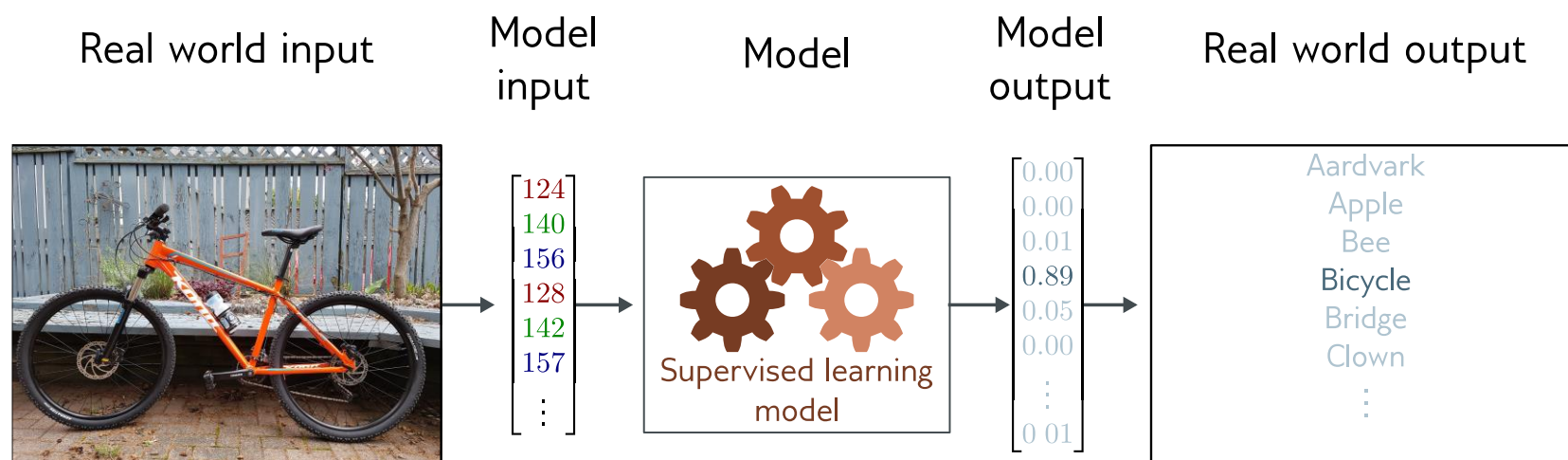
- Múltiple:

¿Qué **CONCEPTO** hay tras una imagen/texto/audio ?

- Regresión:**

- Simple:

- Múltiple:

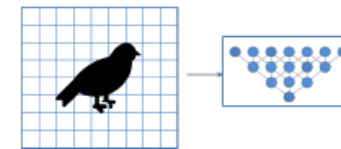


A cada entrada, el modelo decide UNA entre C posibles respuestas (Clases)

Tipos de Tareas

- **Clasificación:**

CLASIFICACIÓN SIMPLE



- Simple:

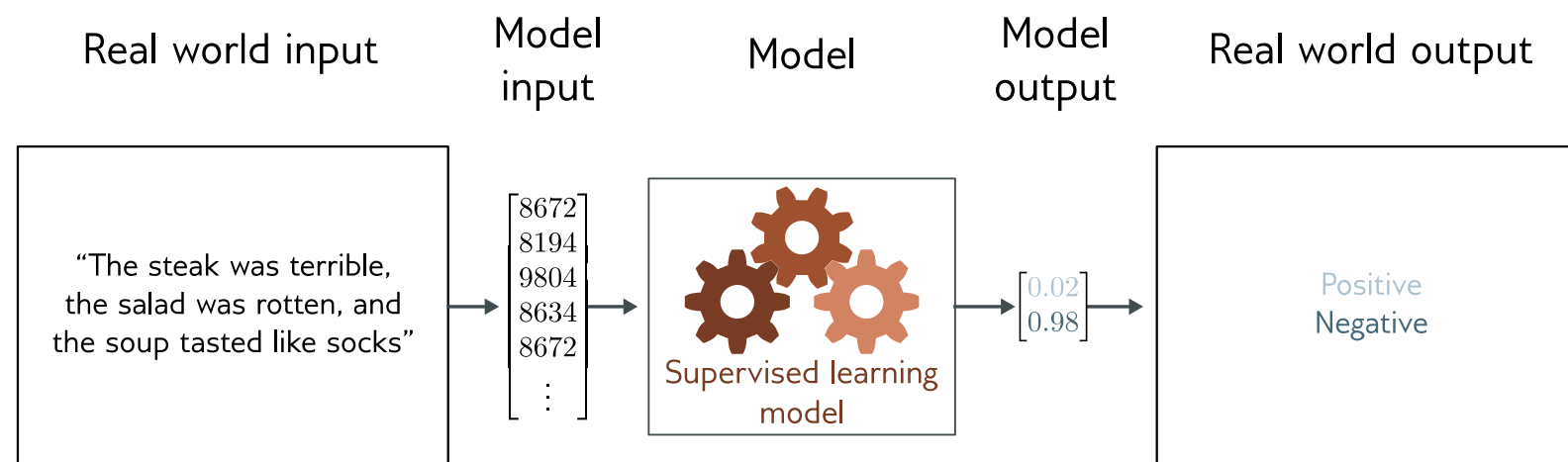
- Múltiple:

¿Qué **CONCEPTO** hay tras una imagen/texto/audio ?

- **Regresión:**

- Simple:

- Múltiple:



En muchas ocasiones **C=2** (Caso binario SI/NO)

Tipos de Tareas

- **Clasificación:**

CLASIFICACIÓN MÚLTIPLE

- Simple:

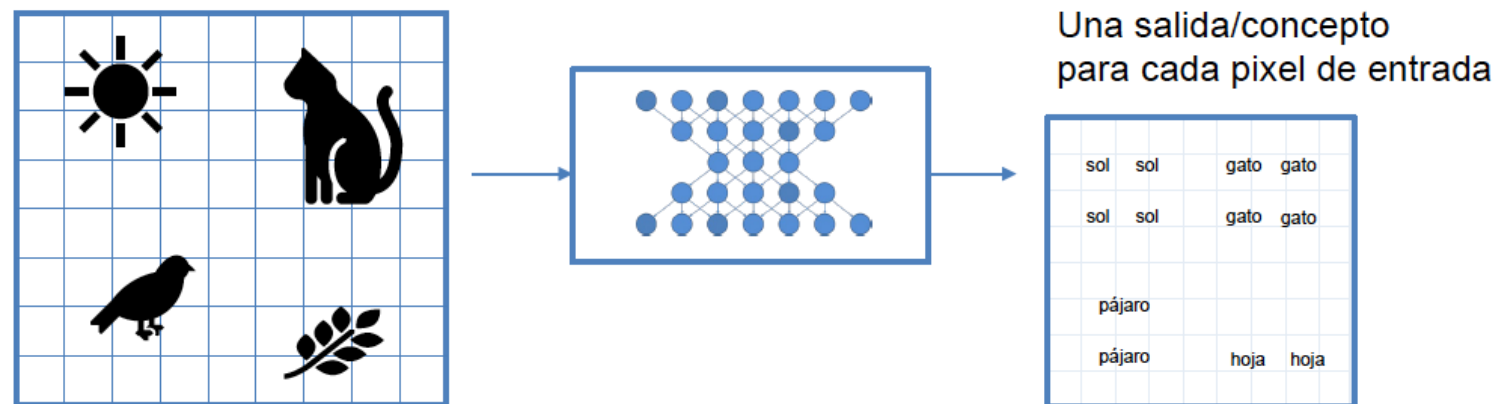
- **Múltiple:**

¿Qué CONCEPTO hay en cada zona/pixel/fragmento de una imagen/texto/audio ?

- **Regresión:**

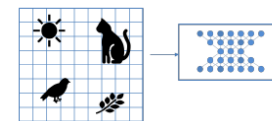
- Simple:

- Múltiple:



Enunciar VARIAS PROPIEDADES/CONCEPTOS de una zona/pixel/fragmento de una imagen/texto/audio ?

Tipos de Tareas



- **Clasificación:**

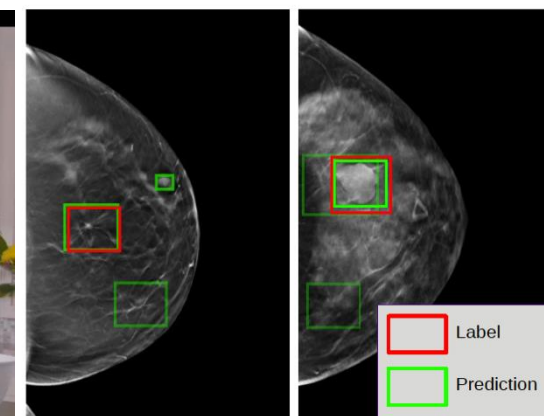
CLASIFICACIÓN MÚLTIPLE

- Simple: **¿Qué **CONCEPTO** hay en cada zona/píxel/fragmento de una imagen/texto/audio ?**

- **Múltiple:**

- **Regresión:**

- Simple:
- Múltiple:



DBTex Challenge:

SPIE-AAPM-NCI DAIR Digital Breast
Tomosynthesis Lesion Detection Challenge



Enunciar **VARIAS PROPIEDADES/CONCEPTOS de una zona/píxel/fragmento de una imagen/texto/audio ?**

Tipos de Tareas

- **Clasificación:**

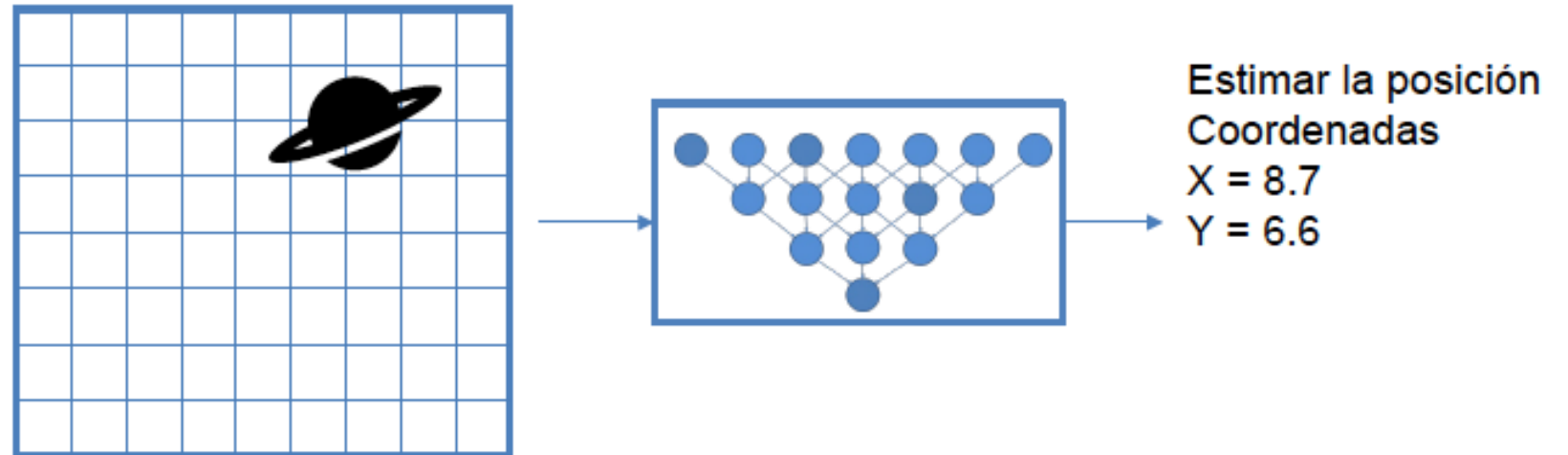
- Simple:
- Múltiple:

REGRESIÓN SIMPLE

Realizar una **PREDICCIÓN NUMÉRICA a partir de una imagen/texto/audio ?**

- **Regresión:**

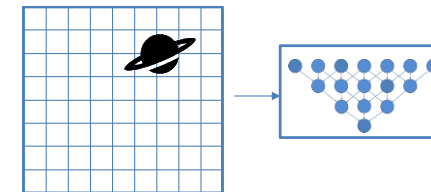
- Simple:
- Múltiple:



Tipos de Tareas

- **Clasificación:**

REGRESIÓN SIMPLE

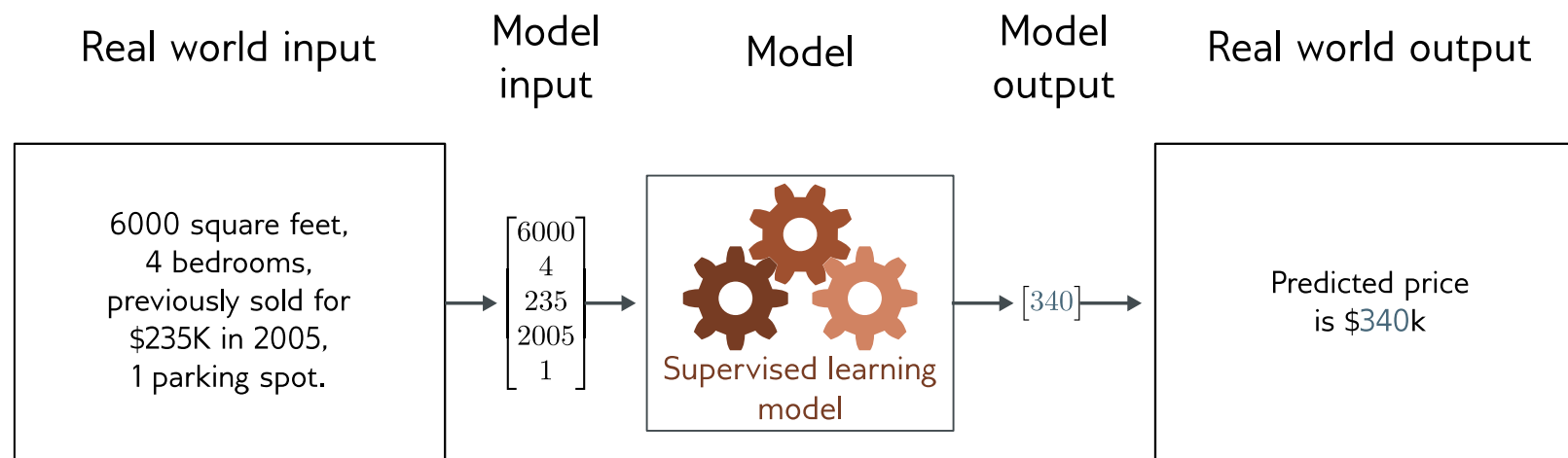


- Simple:
- Múltiple:

Realizar una **PREDICCIÓN NUMÉRICA a partir de una imagen/texto/audio ?**

- **Regresión:**

- Simple:
- Múltiple:



Ante cada entrada, el modelo ofrece **UNA respuesta**

Tipos de Tareas

- **Clasificación:**

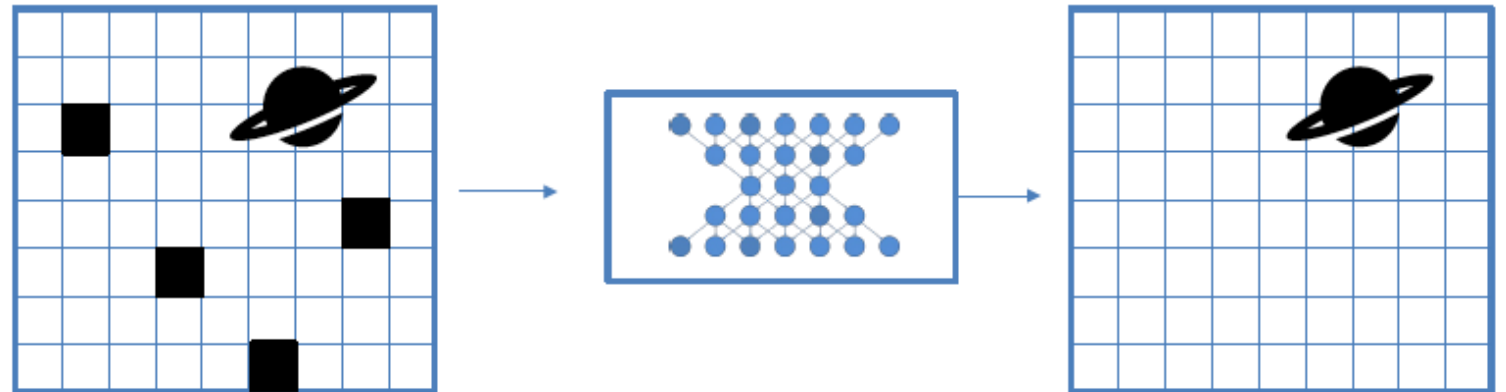
- Simple:
- Múltiple:

REGRESIÓN MÚLTIPLE

PREDECIR VARIOS VALORES NUMÉRICOS para cada zona/pixel/fragmento de una entrada

- **Regresión:**

- Simple:
- Múltiple:



MÉTRICAS DE EVALUACIÓN

- Clasificación:
 - Caso Binario

T = P+N		VALOR REAL	
		POSITIVO (P = TP + FN)	NEGATIVO (N = FN + TN)
VALOR PREDICHO	POSITIVO	TRUE POSITIVE (TP)	FALSE POSITIVE (FN) TYPE II ERROR FALSA ALARMA FALSA ACEPTACIÓN
	NEGATIVO	FALSE NEGATIVE (FN) TYPE I ERROR FALSO RECHAZO MISS	TRUE NEGATIVE (TN)

MÉTRICAS DE EVALUACIÓN

• Clasificación:

- Caso Binario:

- **Accuracy:** $ACC = \frac{TP+TN}{P+N}$

T = P+N		VALOR REAL	
		POSITIVO (P = TP + FN)	NEGATIVO (N = FN + TN)
VALOR PREDICHO	POSITIVO	TRUE POSITIVE (TP)	FALSE POSITIVE (FN) TYPE II ERROR FALSA ALARMAFALSA ACEPTACIÓN
	NEGATIVO	FALSE NEGATIVE (FN) TYPE I ERROR FALSO RECHAZO MISS	TRUE NEGATIVE (TN)

- **Sensitivity / Recall / True Positive Rate:** $TPR = \frac{TP}{P}$
 - También como complementario a **Miss rate/ False Negative Rate:** $TPR = 1 - FNR$
- **Especificidad / True Negative Rate:** $TNR = \frac{TN}{N}$
 - También como complementario a **False Alarm Rate / False Positive Rate:** $TNR = 1 - FPR$

POR COLUMNAS

- **Valor Predictivo Positivo / Precisión:** $PPV = \frac{TP}{TP+FP}$
- **Valor Predictivo Negativo:** $NPV = \frac{TN}{TN+FN}$

POR FILAS

Tipos de Tareas

- Clasificación:

- Simple:

- Múltiple:

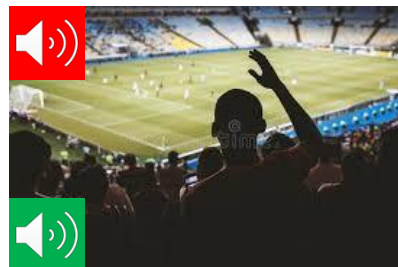
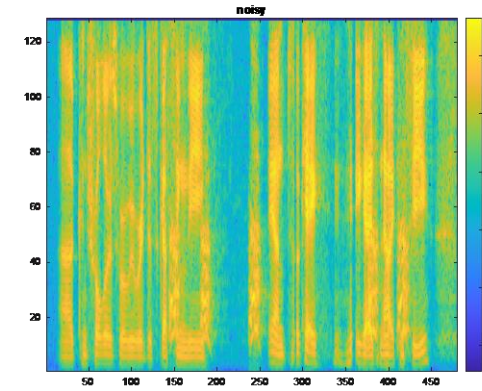
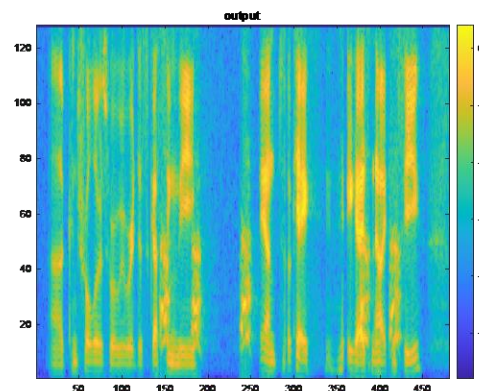
REGRESIÓN MÚLTIPLE

PREDECIR VARIOS VALORES NUMÉRICOS para cada zona/pixel/fragmento de una entrada

- Regresión:

- Simple:

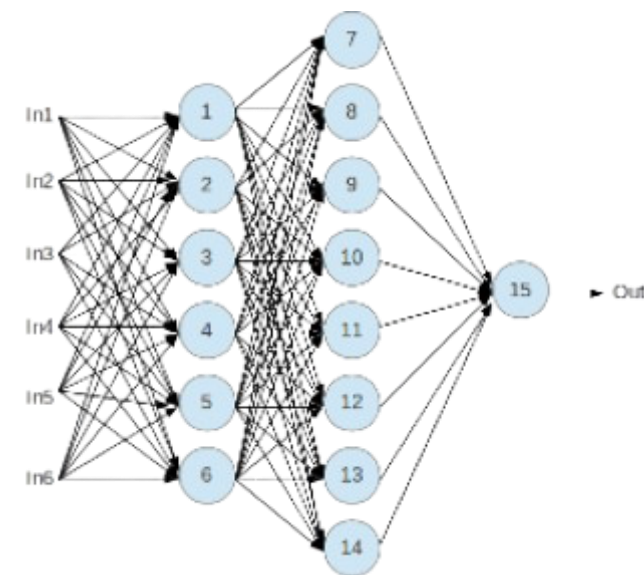
- Múltiple:



Elimina el ruido de un audio ruidoso

Redes Neuronales Artificiales

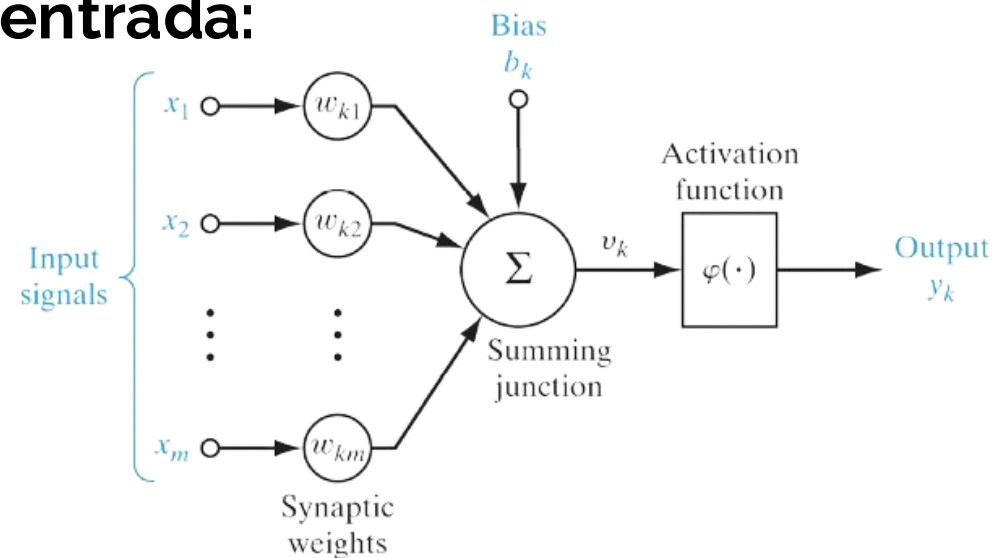
- **Conjunto de elementos simples de procesamiento interrelacionados:**
 - Distribución del procesamiento de la información
 - Cada unidad recibe información de otras, los agrega y transmite una respuesta a través de una función de activación
- **Cada elemento se conecta con otros**
 - Conexiones Sinápticas:
 - Cada conexión tiene un peso
 - Se ajustan a partir de ejemplos
 - Datos de entrenamiento
 - Algoritmos de aprendizaje



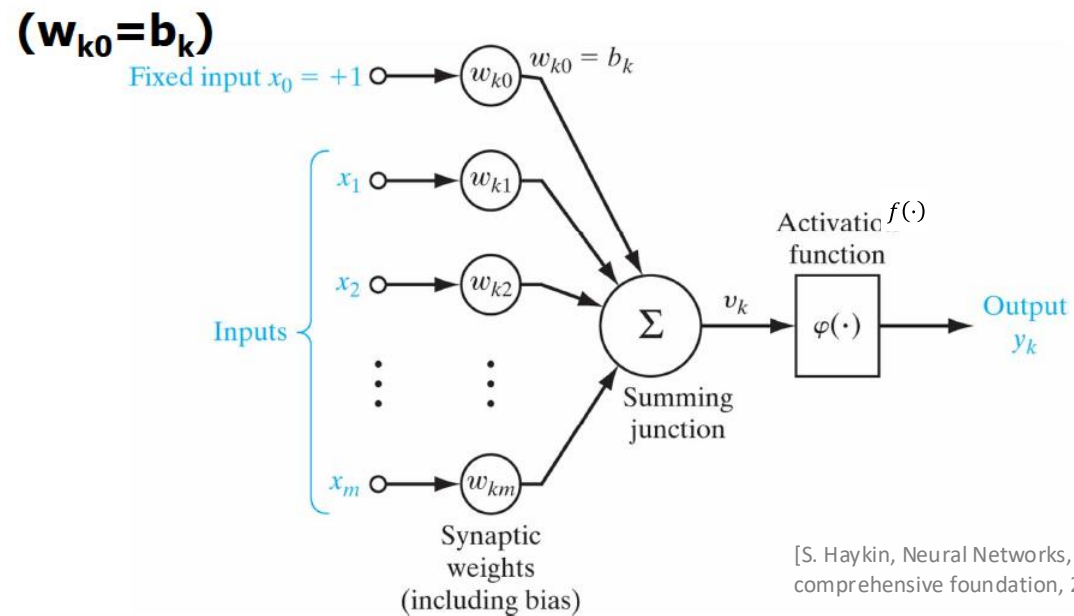
Perceptrón

- Combinación lineal de valores a la entrada:

Vector de pesos $\underline{w}$ + sesgo $\underline{b}$



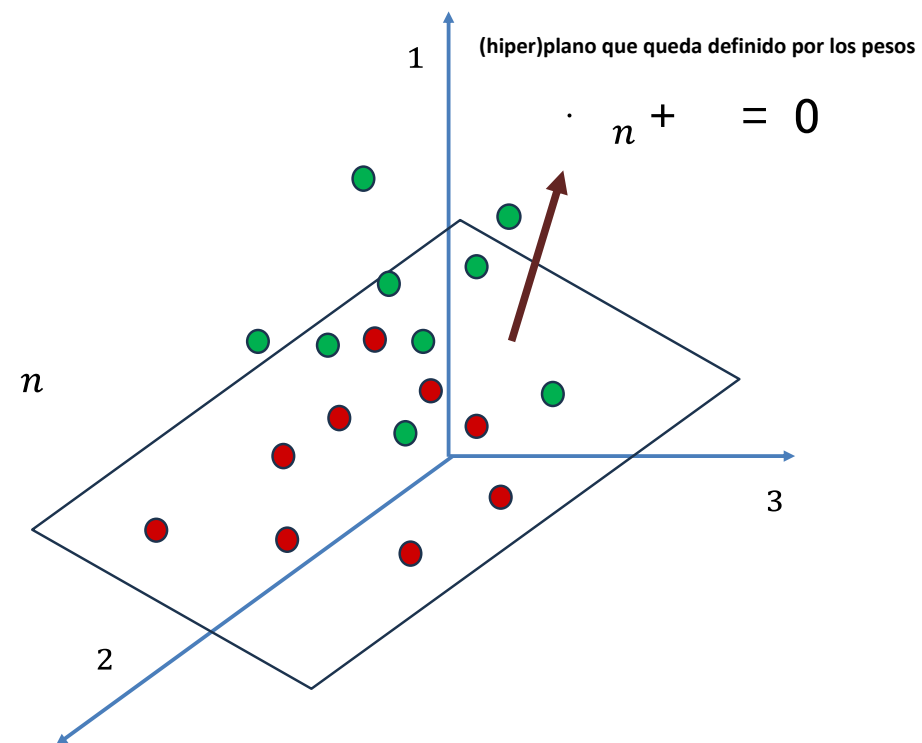
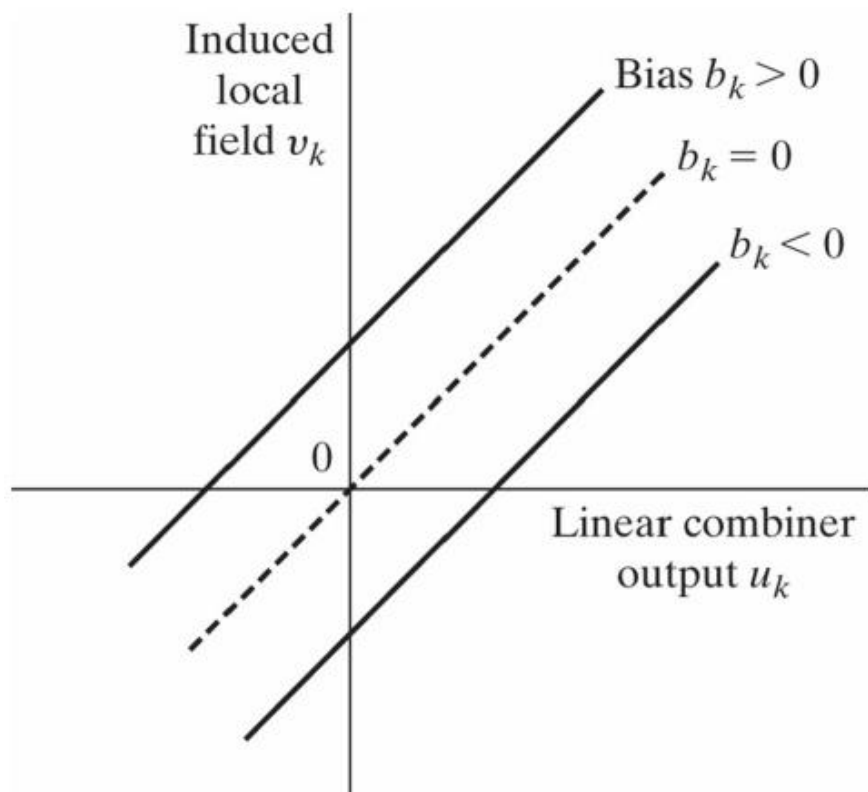
Transformación afín



Función de activación, $f(x_n)$, no lineal

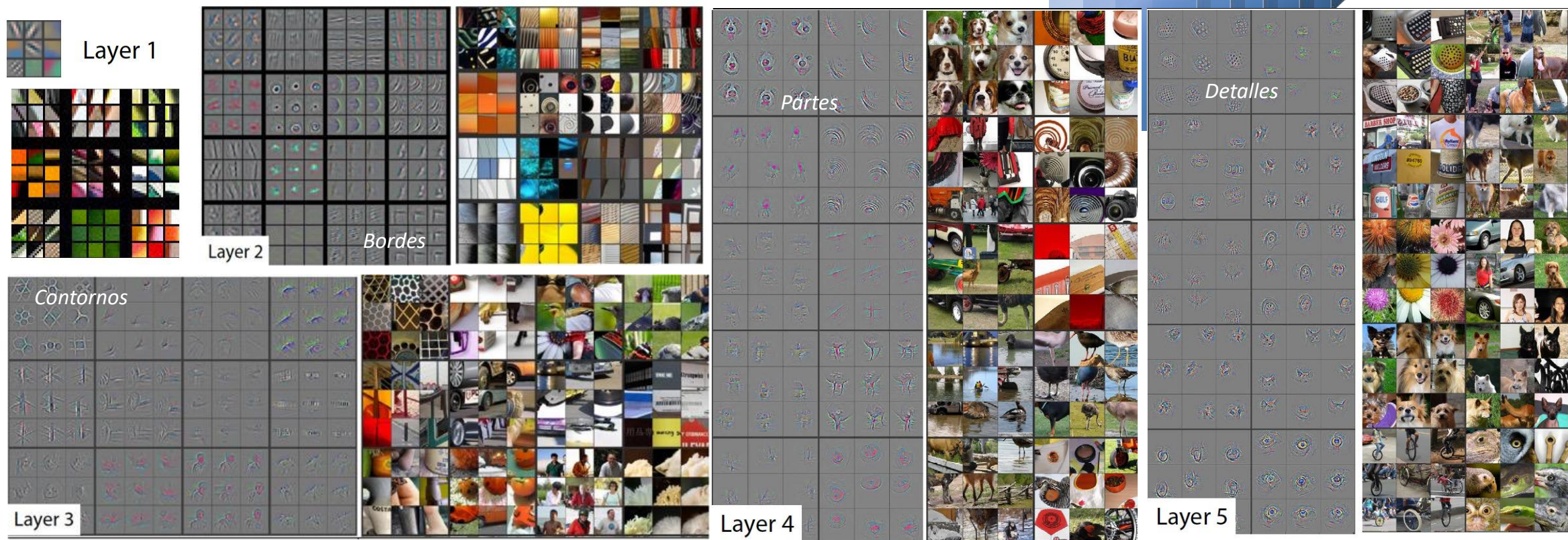
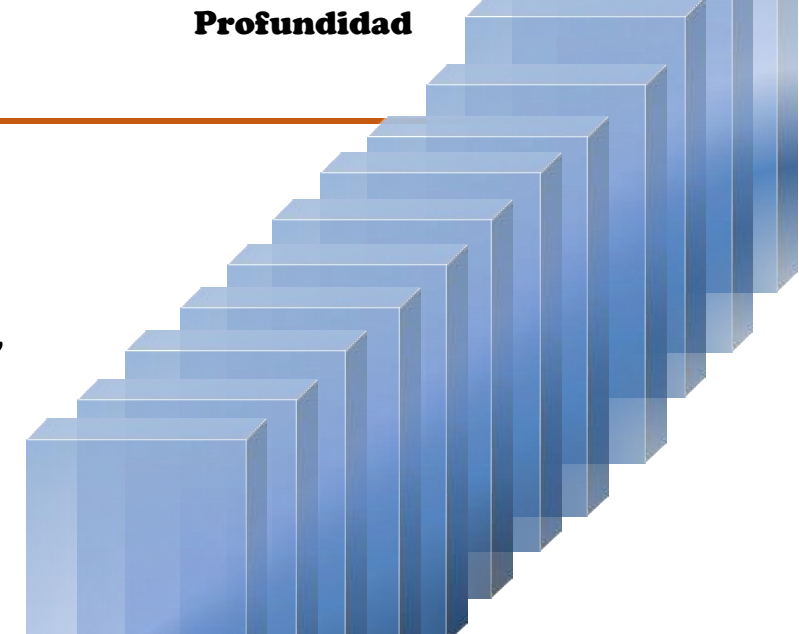
$$o_n = f(\mathbf{x}_n) = \begin{cases} +1, & \mathbf{w} \cdot \mathbf{x}_n + b > 0 \\ -1, & \mathbf{w} \cdot \mathbf{x}_n + b \leq 0 \end{cases}$$

Perceptrón



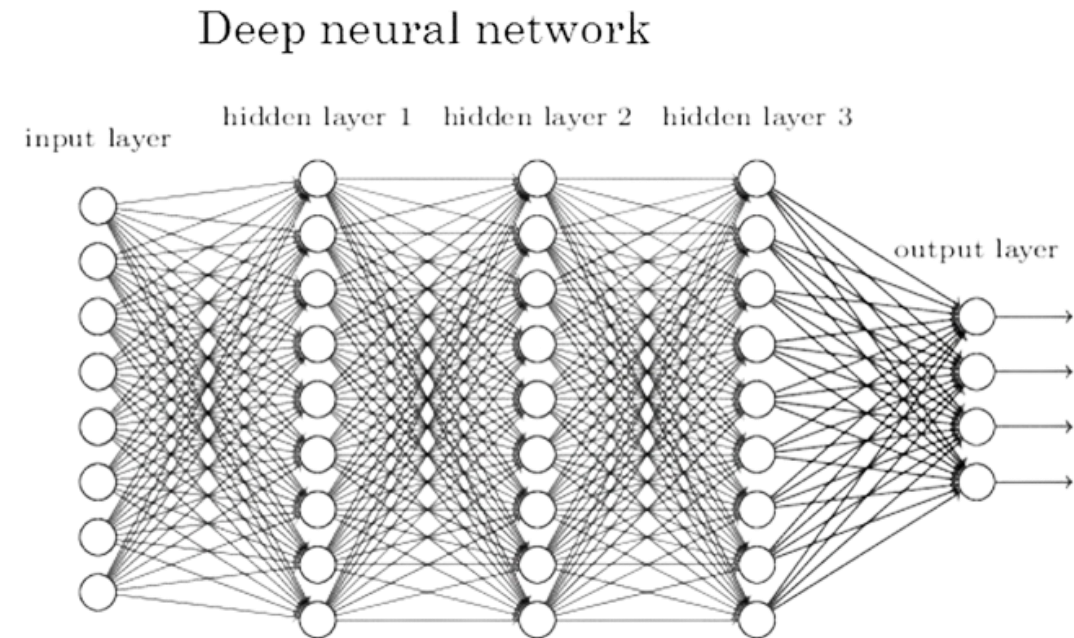
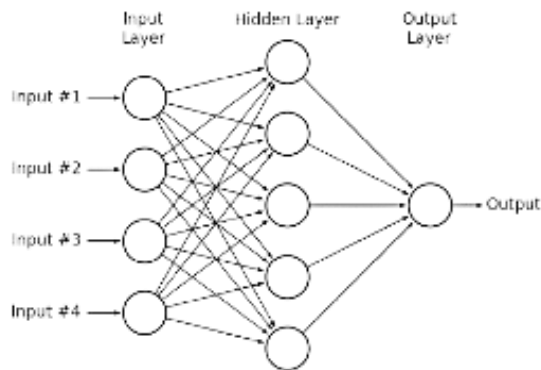
Redes Neuronales Artificiales

- **Procesado en capas:**
 - Aprenden diferentes niveles de representación de los datos, desde lo más simple, hasta lo más abstracto

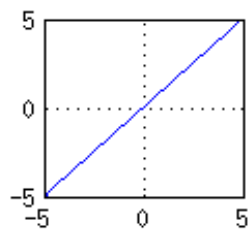


Multilayer Perceptron / Deep Feedforward Network

- **Múltiples Neuronas (perceptrón) básicas organizadas en capas**
 - **Arquitectura de la red:**
 - **Capa de entrada:** Toma los datos de entrada (en forma vector)
 - Conviene acondicionar los datos: Normalización
 - **Capas Ocultas:** Sin contacto ni con la entrada ni con la salida
 - Agrupación de perceptrones
 - **Capas de Salida:**
 - Resultado del proceso

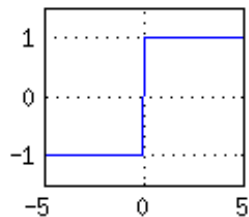


Funciones de Activación



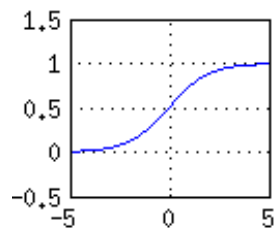
Linear

$$f(x) = x$$



Sign

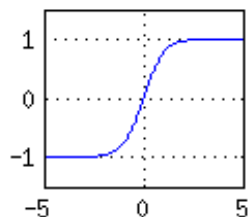
$$f(x) = \begin{cases} +1, & x \geq 0 \\ -1, & x < 0 \end{cases}$$



Logistic

$$f(x) = \frac{1}{1 + \exp(-x)}$$

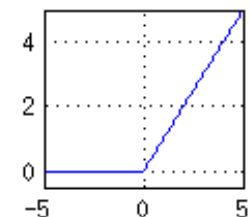
$$f'(x) = f(x)(1 - f(x))$$



Tanh

$$f(x) = \tanh(x)$$

$$f'(x) = (1 - f(x)^2)$$



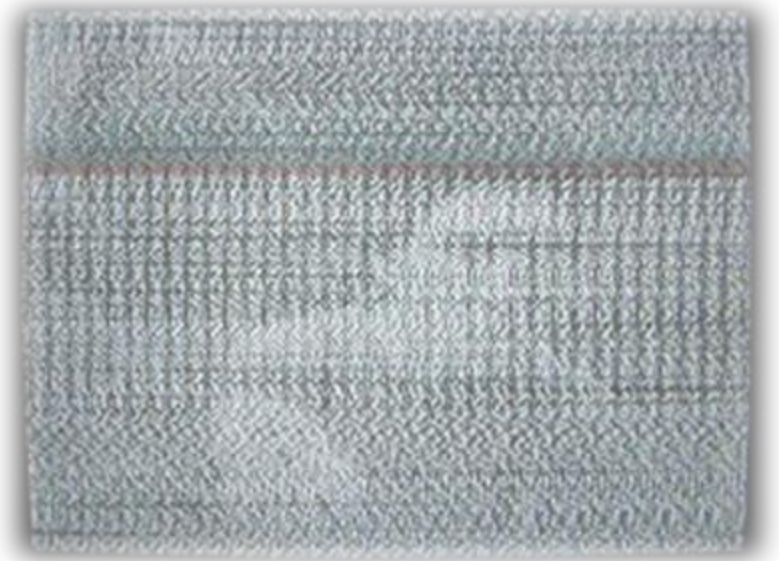
ReLu

$$f(x) = \max(0, x)$$

$$f'(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$$

Entrenamiento / Aprendizaje

- **Ajuste de los parámetros del modelo:**
 - Selección de los datos de entrenamiento
 - Exposición de los datos a la red
 - Observación de la salida
 - Variar los pesos para hacer disminuir el error



Deep Forward Networks

- **Tipos de datos:**
 - **Etiquetas:**
 - Son vectores objetivo: y_n
 - Si la red actúa como regresor (prediciendo valores):
 - $y_n \in \mathbb{R}^D$
 - Si la red actúa como un clasificador
 - $y_n \in \{0, 1\}^C$ (C clases) One-hot encoding
 - $y_n \in \{0, 1, \dots, C - 1\}$ (C clases)
 - **Salidas:**
 - Son el producto de la red: y_n
 - Tendrán la misma dimensión y serán del mismo tipo que las etiquetas
 - **Entradas:**
 - Son los datos que alimentan la red: x_n

Deep Forward Networks

- Entrenamiento:

- Pares de datos (x_n, y_n) , $x_n \in \mathbb{R}^D$; $y_n \in \mathbb{R}^D$ ó $y_n \in \{0, 1\}^C$...

- Función de coste $J(X, Y, \Theta)$:

- Es una función de los ejemplos X , de las etiquetas Y y de los parámetros del modelo Θ

- Error cuadrático:

- $$J = \sum_n (y_n - o_n)^2$$

- Errores de Clasificación

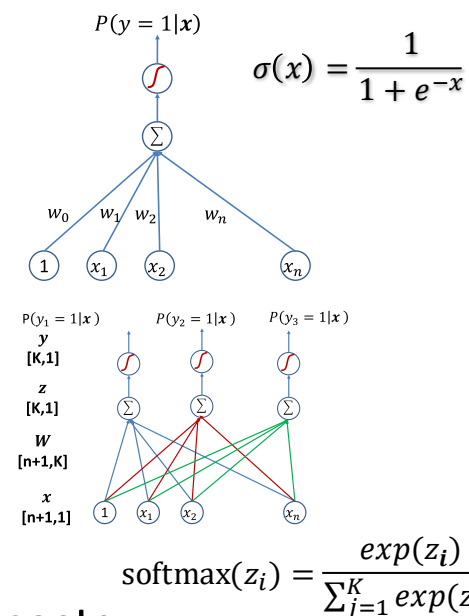
- $$J(\hat{y}, y) = \sum_n L_{CE}(\hat{y}_n, y_n)$$

Caso Binario

$$L_{CE}(\hat{y}_n, y_n) = -y_n \log y_n - (1 - y_n) \log(1 - \hat{y}_n) = -\log p(y_n | x_n)$$

Caso Multiclase

$$L_{CE}(\hat{y}_n, y_n) = -\sum_{k=1}^K y_n(k) \log \hat{y}_n(k)$$



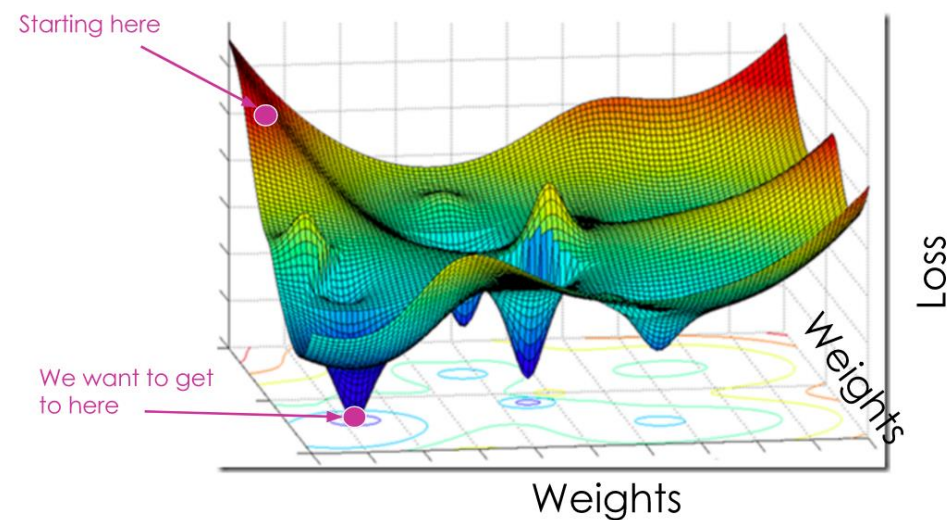
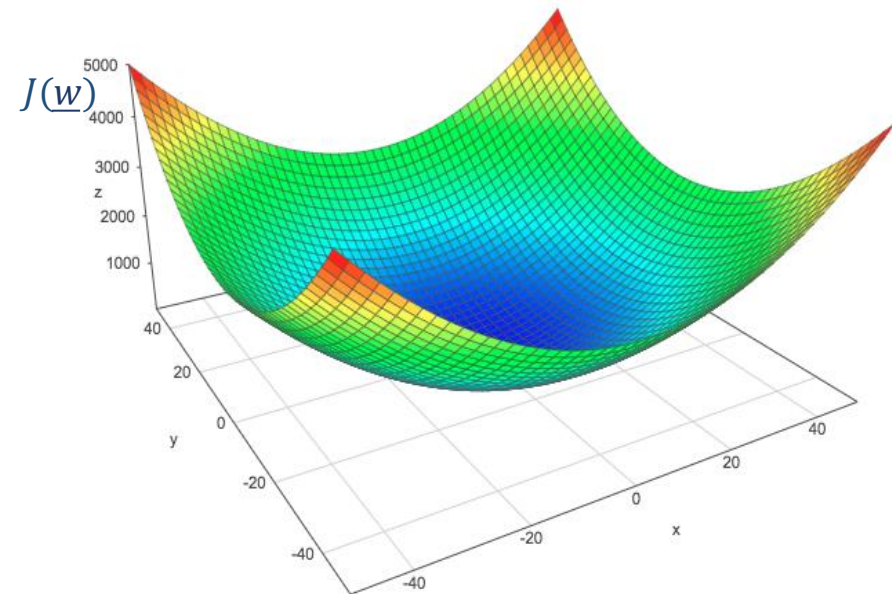
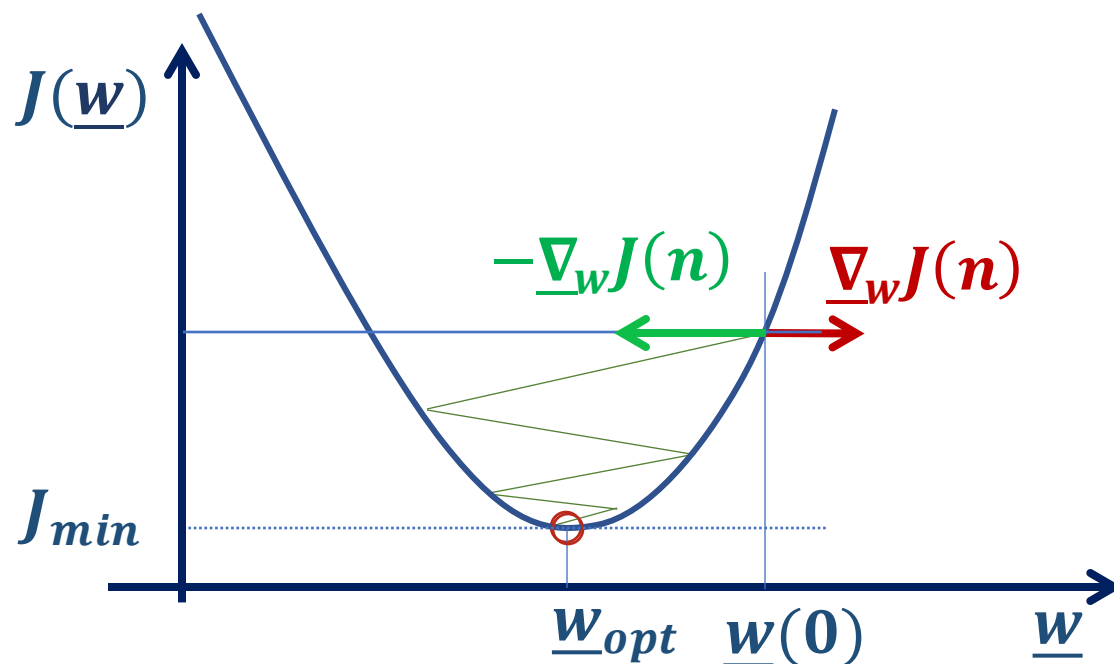
- Objetivo:

- Encontrar los parámetros del modelo que minimicen la función de coste:

$$\{\hat{w}, \hat{b}\} = \arg \min_{w, b} J(X, Y, w, b)$$

Entrenamiento

- **Gradient Based Optimization:**
 - Superficie de error compleja
 - Múltiples mínimos locales
 - No hay garantía de llegar al mínimo global



Entrenamiento

- **BACKPROPAGATION:**
 - Método para implementar el aprendizaje basado en gradiente (GRADIENT BASED)

- **Red Neuronal:**

$$o_n = f(w, x)$$

- **Función de coste:**

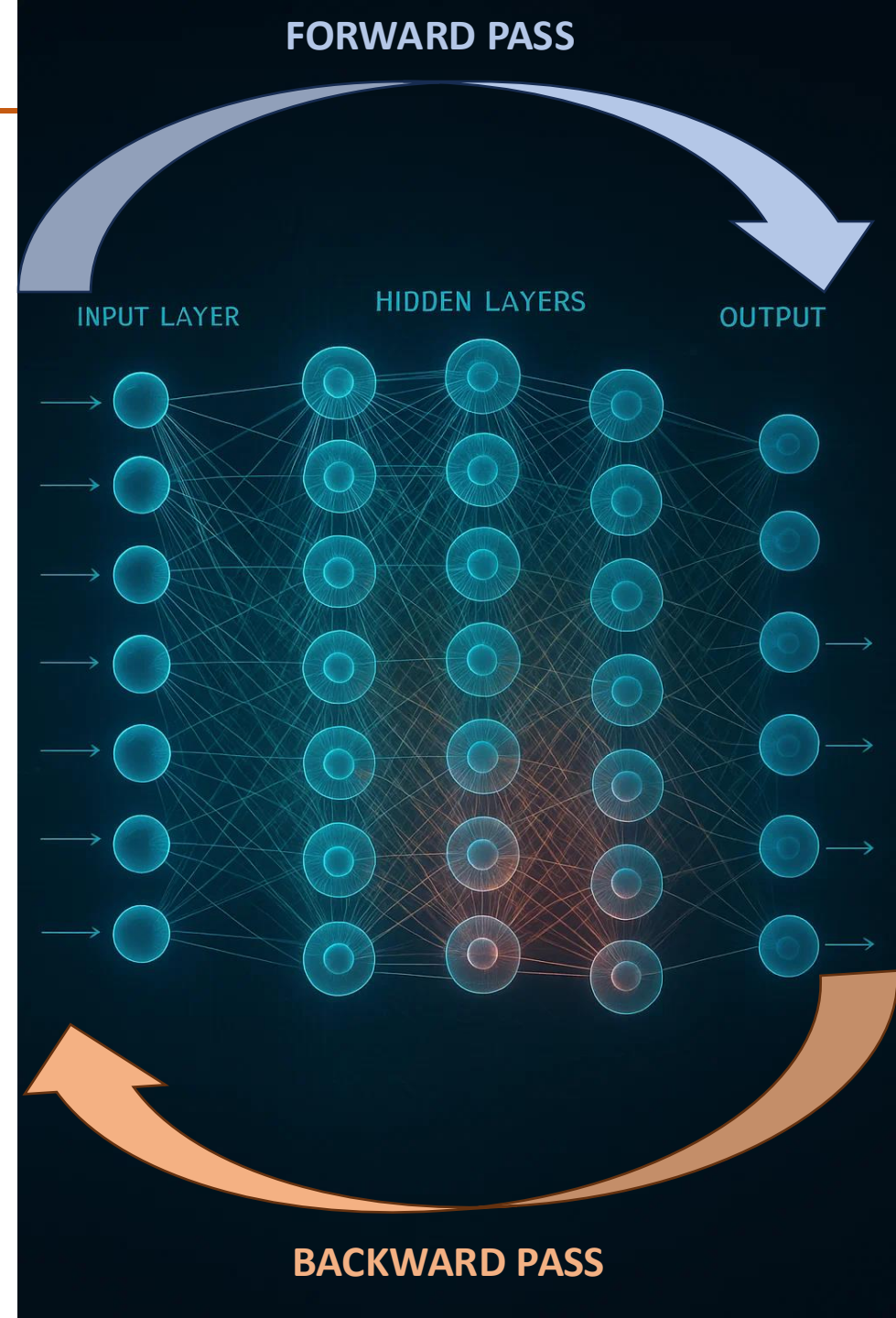
$$J(X, Y, \theta)$$

- **Idea Básica:**

Calcular de forma eficiente todas las **derivadas parciales** de la función de coste, $J(X, Y, \theta)$, respecto de todos los parámetros ajustables de la función, sus pesos w , para un valor dado de la entrada x

$$J = \sum_n J_n(o_n, y_n)$$

$$\frac{\partial J}{\partial w_i} = \sum_n \frac{\partial J_n}{\partial o_n} \frac{\partial o_n}{\partial w_i}$$



Entrenamiento

- Los pesos se ajustan siguiendo la **dirección contraria al gradiente**

$$\mathbf{w}_i^{(j+1)} = \mathbf{w}_i^{(j)} - \eta \frac{\partial J}{\partial \mathbf{w}_i} = \mathbf{w}_i^{(j)} - \eta \sum_n \frac{\partial J_n}{\partial o_n} \frac{\partial o_n}{\partial \mathbf{w}_i}$$

- Este proceso se realiza de forma iterativa utilizando los datos de entrenamiento
- η es lo que se conoce como tasa de aprendizaje o **learning rate**:
 - Valores grandes: aprendizaje más rápido y menos preciso.
 - Valores pequeños: aprendizaje más lento, pero más preciso

Entrenamiento

- Es necesario conocer el gradiente para todas y cada una de las capas
- Esta información debe fluir desde la capa de salida hacia la capa de entrada.
- Obtener la expresión analítica del gradiente es posible (sencillo) pero computacionalmente caro
 - **Backpropagation** hace esto computacionalmente eficiente, aplicando la regla de la cadena (chain rule)

Si f y g son funciones reales de variable real: $y = g(x)$ $z = f(y) = f(g(x))$ $\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x}$

Si $\underline{x} \in \mathbb{R}^m$; $\underline{y} \in \mathbb{R}^n$; $\underline{g}$ mapea de $\mathbb{R}^m$ a $\mathbb{R}^n$, y f mapea de $\mathbb{R}^n$ a $\mathbb{R}$
 $\underline{y} = \underline{g}(\underline{x})$ $z = f(\underline{y}) = f(\underline{g}(\underline{x}))$

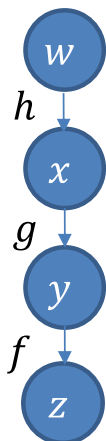
$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}$$

Computational Graphs

- ¿Como llevar la información del gradiente a todos y cada uno de los pesos de forma eficiente?
 - Se crean grafos de las expresiones involucradas en el cálculo de las salidas
 - Estos grafos indican qué depende de qué y cómo

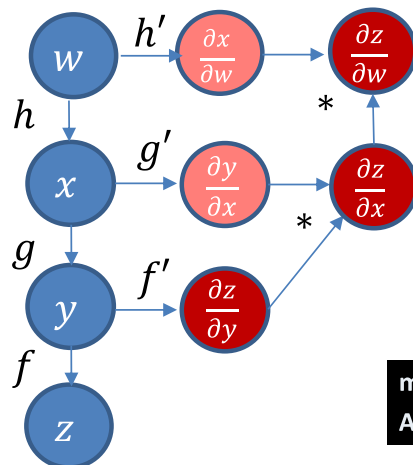
Forward Pass

$$z = f(y) = f(g(x)) = f(g(h(w)))$$



Backward Pass

$$\frac{\partial z}{\partial w} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \frac{\partial x}{\partial w}$$



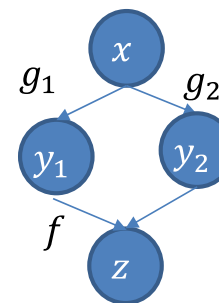
micrograd
Andrej Karpathy

Forward Pass

$$z = f(y_1, y_2);$$

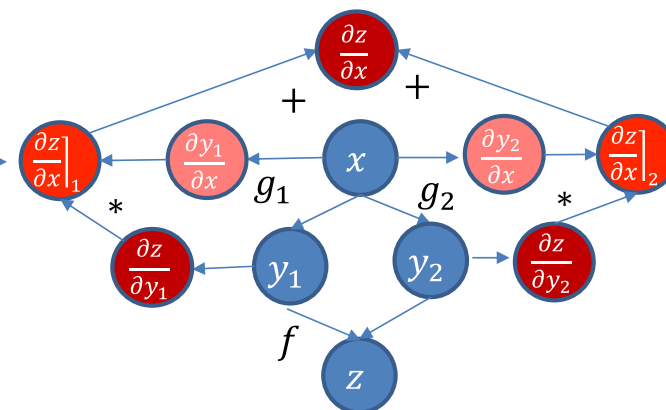
$$y_1 = g_1(x)$$

$$y_2 = g_2(x)$$

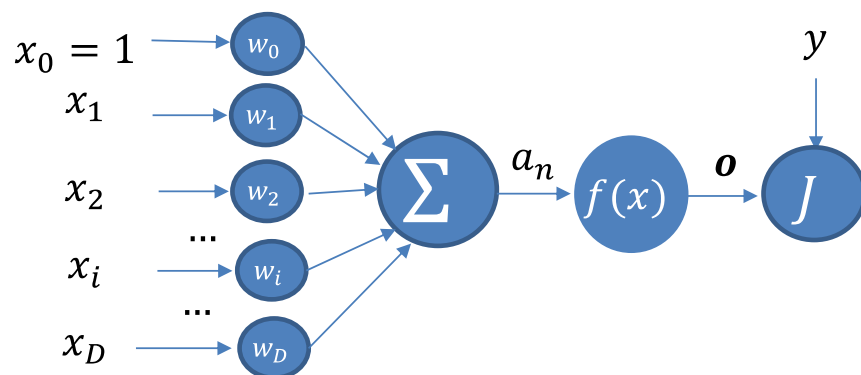


Backward Pass

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1} \frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2} \frac{\partial y_2}{\partial x}$$



Perceptrón con función de coste MSE:



Forward Pass

$$a_n = \sum_{i=0}^D w_i x_i = \mathbf{w}^T \mathbf{x}_n$$

$$o_n = f(a_n)$$

$$e_n = y_n - o_n$$

$$J = \frac{1}{2} (e_n)^2$$

Backward Pass

$$\nabla_{w_i}(J) = \frac{\partial J}{\partial w_i} = \frac{\partial J}{\partial o} \frac{\partial o}{\partial a} \frac{\partial a}{\partial w_i}$$

$$\frac{\partial a}{\partial w_i} = x_i \quad i = 1, 2, \dots, D$$

$$\frac{\partial a}{\partial w_0} = x_0 = 1$$

$$\frac{\partial o}{\partial a} = f'(a) = f'(\mathbf{w}^T \mathbf{x}_n)$$

$$\frac{\partial J}{\partial o} = -(y - o) = -e$$

$$\frac{\partial J}{\partial w_i} = -e \cdot f'(\mathbf{w}^T \mathbf{x}_n) \cdot x_i$$



$$w_i^{(j+1)} = w_i^{(j)} - \eta \frac{\partial J}{\partial w_i} = w_i^{(j)} + \eta \cdot e \cdot f'(\mathbf{w}^T \mathbf{x}_n) \cdot x_i$$

Stochastic Gradient Descent - SGD

- El gradiente no se calcula haciendo uso de toda la base de datos de entrenamiento sino por bloques y además en orden aleatorio en cada iteración
- SGD converge a mínimos locales. Como la función de coste ya no es tan simple como en el caso lineal, el mínimo local no tiene por qué ser global

- Gradiente Instantáneo:

- Actualización tras cada ejemplo de entrenamiento $w_i(n+1) = w_i(n) - \eta \frac{\partial \mathcal{J}(\mathbf{x}_n, y_n, \mathbf{w}^{(n)})}{\partial w_i}$

- Gradiente por lotes, (Batch Gradient):

- Actualización tras cada bloque k $w_i(k+1) = w_i(k) - \eta \frac{\partial \sum_{n \in k} \mathcal{J}(\mathbf{x}_n, y_n, \mathbf{w}^{(k)})}{\partial w_i}$

- Gradiente Global:

- Cálculo para toda la base de datos de entrenamiento.
- Actualización tras cada iteración (epoch) de toda la base de datos.
- Alto coste computacional, robusto y baja velocidad de convergencia $w_i^{(j+1)} = w_i^{(j)} - \alpha \frac{\partial \sum_n \mathcal{J}(\mathbf{x}_n, y_n, \mathbf{w}^{(j)})}{\partial w_i}$

Entrenamiento

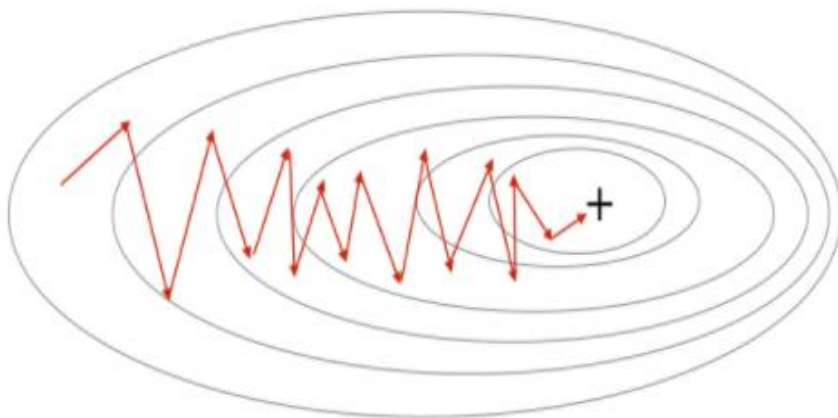
- Ejemplo de código del entrenamiento de una red neuronal:

```
def train_model(model, optimizer, criterion, train_loader, val_loader,
                num_epochs=10, checkpoint_dir=None, device='cpu'):
    # Cargar el checkpoint más reciente
    start_epoch, model, optimizer = load_checkpoint(model, optimizer, checkpoint_dir)
    # Mover el modelo a la GPU/CPU
    model.to(device)
    # Iniciar el loop de entrenamiento
    for epoch in range(start_epoch, num_epochs):
        # Poner el modelo en modo de entrenamiento
        model.train()
        # Inicializar la variable para almacenar la función de coste
        train_loss = 0
        # Iterar sobre los batches del dataloader de entrenamiento
        for batch_number, (batch_data, batch_labels) in enumerate(train_loader):
            # Reiniciar los gradientes
            optimizer.zero_grad()
            # Mover los datos y las etiquetas a la GPU/C
            batch_data = batch_data.to(device)
            batch_labels = batch_labels.to(device)
            # Forward pass
            outputs = model(batch_data)
            # Calcular el valor de la función de coste
            loss = criterion(outputs, batch_labels)
            # Backward pass
            loss.backward()
            # Actualizar los pesos
            optimizer.step()
```

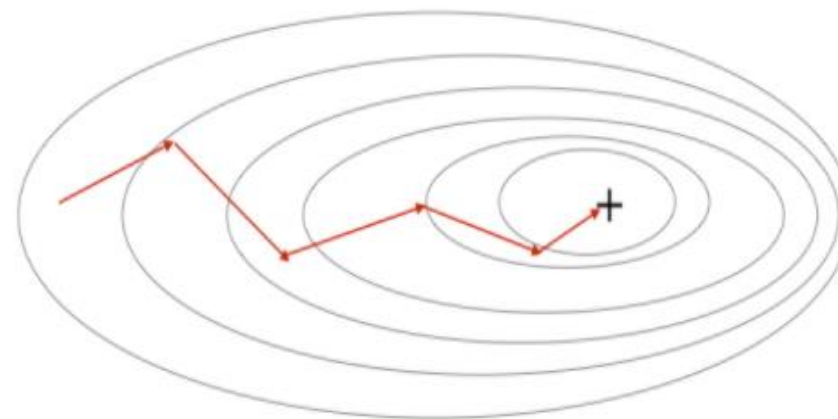
Backpropagation: Implicaciones y orientaciones prácticas

- **Mini-Batch processing:**
 - Mejor resultado, uso de “pequeños” lotes: **mini-batches**
 - Se puede optimizar mucho con el uso de GPU
 - Es conveniente aleatorizar los datos de entrenamiento
 - Uso de ejemplos más difíciles

Stochastic Gradient Descent



Mini-Batch Gradient Descent



Implicaciones y orientaciones prácticas

- **Normalización:**
 - Acondicionamiento de las entradas (y valores intermedios)
 - Conveniente para un buen funcionamiento
 - Eliminar media y aproximar varianza de todas las dimensiones
 - Decorrelación de las diferentes variables en juego

- Z-NORM

$$\underline{x'} = \frac{\underline{x} - \underline{\mu}_x}{\sigma_x}$$

MIN-MAX

$$\underline{x'} = \frac{\underline{x} - \underline{x}_{min}}{\underline{x}_{max} - \underline{x}_{min}}$$

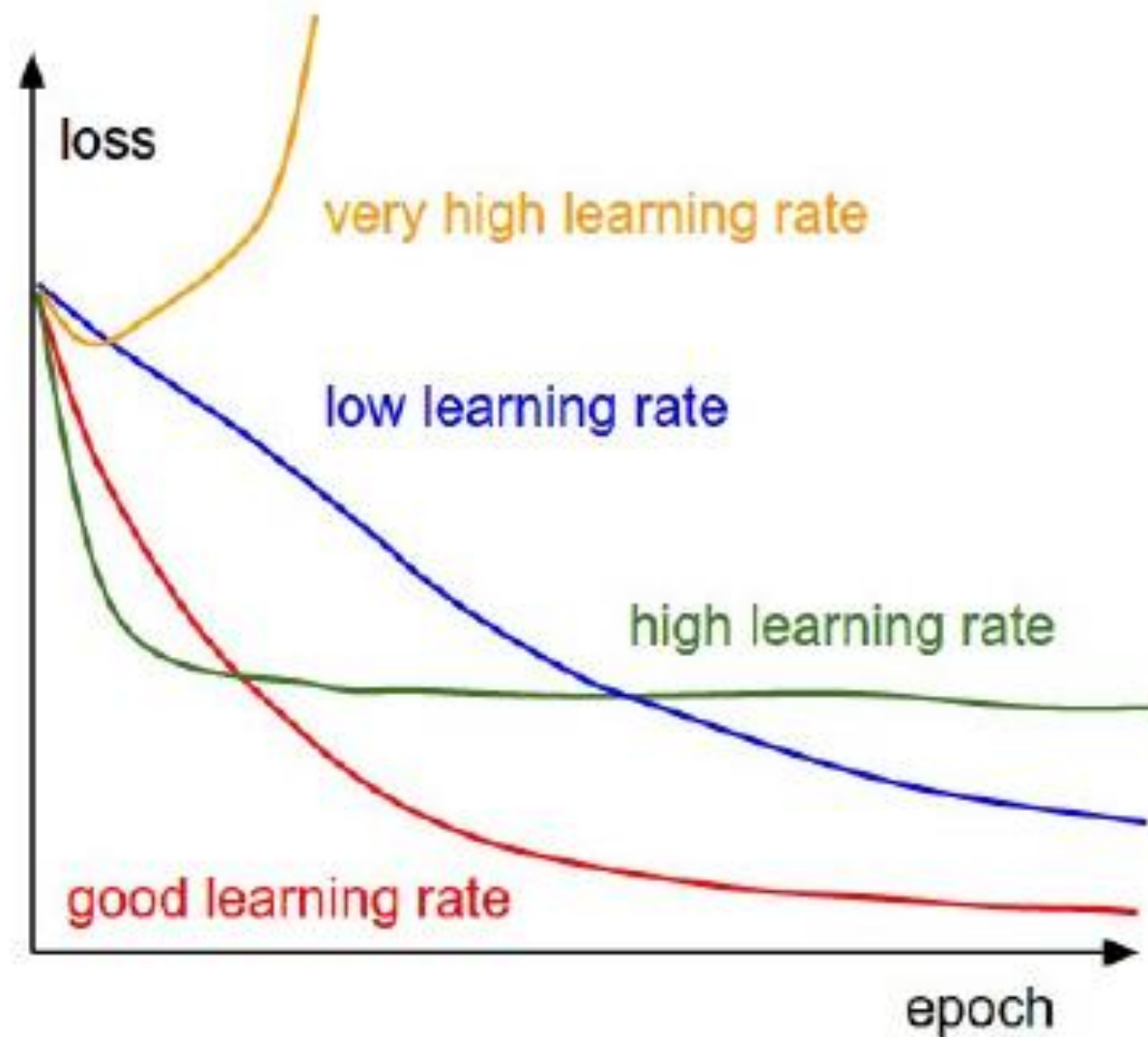
Implicaciones y orientaciones prácticas

- **INICIALIZACIÓN:**
 - Aleatorizar el valor inicial de los pesos
 - Hay que **“romper la simetría”**
 - Distintas unidades conectadas a las mismas entradas inicializadas con el mismo valor tenderán a aprender lo mismo.
 - Distribución uniforme con media 0 y desviación estándar dependiente de las conexiones que llegan al nodo.
 - pequeñas variaciones en los pesos pueden tener alta repercusión si hay muchas conexiones (fan-in alto)
- **Convergencia vs Generalización:**
 - Buena convergencia, llegada a un punto estable rápido
 - Buena generalización, buen funcionamiento para datos no vistos durante el entrenamiento y con propiedades ligeramente distintas
- Algunas inicializaciones son buenas para una rápida convergencia, pero malas para la generalización y otras al revés.

Implicaciones y orientaciones prácticas

- **Learning Rate:**

- Valor fijo
- Adaptable
- Ajustado a cada conexión

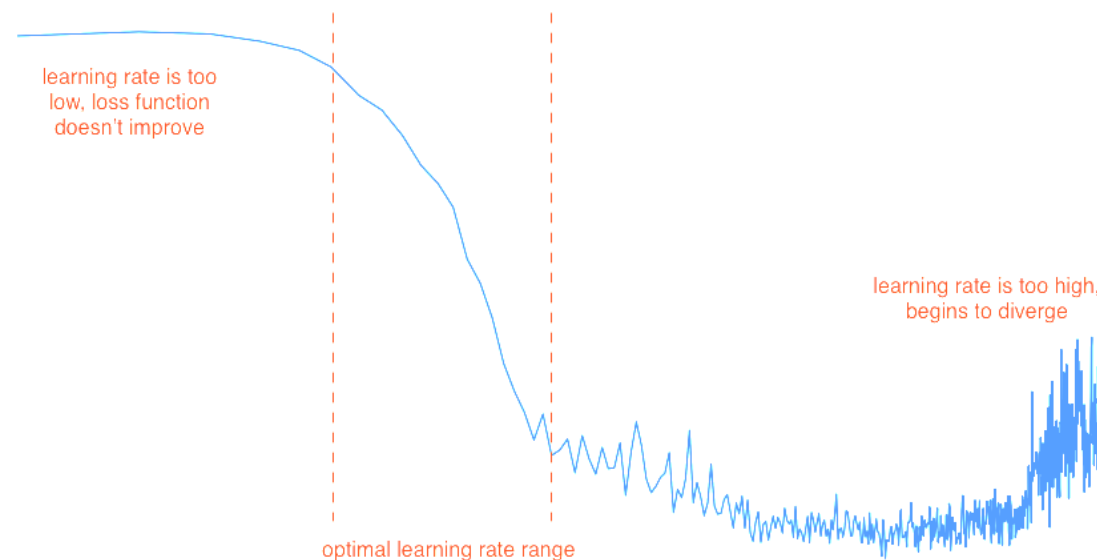


Estrategias de Aceleración de Convergencia

- **Estrategias de aceleración de la convergencia:**
 - Momentos
 - Adaptive Learning Rate
 - AdaGrad
 - RMSprop
 - Adam
 - ...

Estrategias de Aceleración de Convergencia

- Estrategias de aceleración de la convergencia:
 - **Adaptive Learning Rate**
 - Se reduce el learning rate cuando la reducción de la función de coste se ralentiza (plateau)
 - Se suele usar SGD con momentum y decaying learning rate (hasta un valor mínimo del learning rate, con caída exponencial, o dividiéndolo por un factor entre 2 y 10 cuando se detecta la meseta)



Estrategias de Aceleración de Convergencia

- **Adaptive Learning Rate:**

- **AdaGrad**

- Se acumulan los gradientes al cuadrado

$$\mathbf{r}(k) = \mathbf{r}(k - 1) + \left(\frac{\partial \sum_{n \in k} \mathcal{J}(\mathbf{x}_n, \mathbf{y}_n, \mathbf{w}^{(k)})}{\partial \mathbf{w}_i} \right)^2$$

- El learning rate se divide por la raíz cuadrada del acumulado de los gradientes al cuadrado

$$\eta(k) = \frac{\epsilon}{\delta + \sqrt{\mathbf{r}(k)}}$$

Estrategias de Aceleración de Convergencia

- **Adaptive Learning Rate:**

- **RMSProp**

- Modificación sobre AdaGrad
- Cambia la acumulación de gradientes al cuadrado por una media móvil (exponencial)

$$\mathbf{r}(k) = \rho \cdot \mathbf{r}(k - 1) + (1 - \rho) \left(\frac{\partial \sum_{n \in k} \mathcal{J}(\mathbf{x}_n, \mathbf{y}_n, \mathbf{w}^{(k)})}{\partial \mathbf{w}_i} \right)^2$$

- El learning rate se divide por la raíz cuadrada del acumulado de los gradientes al cuadrado

$$\eta(k) = \frac{\epsilon}{\delta + \sqrt{\mathbf{r}(k)}}$$

Estrategias de Aceleración de Convergencia

- **Adaptive Learning Rate:**

- **Adam**

- Añade momento al RMSProp (Adam: “adaptive moments”)

$$s_i(k) = \rho_1 \cdot s_i(k-1) + (1 - \rho_1) \frac{\partial \sum_{n \in k} \mathcal{J}(x_n, y_n, w^{(k)})}{\partial w_i}$$

- Acumulación de gradientes al cuadrado con una media móvil (exponencial) para adaptar Learning rate

$$r(k) = \rho_2 \cdot r(k-1) + (1 - \rho_2) \left(\frac{\partial \sum_{n \in k} \mathcal{J}(x_n, y_n, w^{(k)})}{\partial w_i} \right)^2 \quad \eta(k) = \frac{\epsilon}{\delta + \sqrt{r(k)}}$$

- Combinados

$$w_i(k+1) = w_i(k) - \eta(k) s_i(k)$$

Estrategias de Aceleración de Convergencia

- Momentos
- Adaptive Learning Rate
 - AdaGrad
 - RMSprop
 - Adam
 - ...

- ¿Cuál usar?

No hay consenso claro y depende del problema y de los datos.

Implicaciones y orientaciones prácticas

- **Batch Normalization:**
 - Estrategia para reparametrizar redes profundas
 - Los gradientes y actualizaciones en una capa dependen en gran medida del resto de las capas lo que hace difícil el ajuste de hiperparámetros como el learning rate.
 - Si $\mathbf{H}$, es un minibatch de activaciones de una capa, estas se normalizan para que tengan media nula y varianza unidad

$$H' = \frac{H - \mu}{\sigma} \quad \mu = \frac{1}{m} \sum_i H_i \quad \sigma = \sqrt{\delta + \frac{1}{m} \sum_i (H - \mu)_i^2}$$

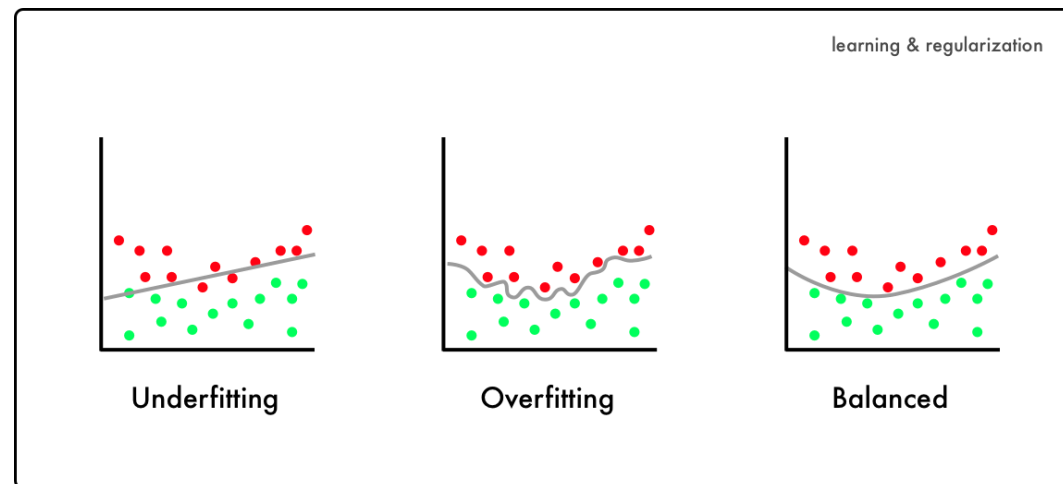
- En test μ y σ pueden tomar valores aprendidos durante el entrenamiento

Implicaciones y orientaciones prácticas

- Técnicas de Generalización y Regularización:

- **Generalización:**

- Si la red está sobredimensionada (demasiados parámetros) se corre el riesgo de aprender las peculiaridades de los datos de entrenamiento:
 - Sobreajuste u overfitting.
- Para evitar el sobreajuste:
 - Contar con más datos:
 - Aumentar los dataset de train
 - **Data augmentation**
 - Limitar la capacidad de la red para no exceder la complejidad del problema que se aborda:
 - Limitar el número de capas
 - **Weight Sharing:** Varias unidades comparten sus pesos



Implicaciones y orientaciones prácticas

- Técnicas de Generalización y Regularización :
 - Estrategias:
 - **Early stopping:**
 - El entrenamiento se detiene antes de que la red llegue a overfitting
 - **Weight decay:**
 - Se intenta que los pesos no alcancen valores muy grandes (norma L2 ó norma L1)
 - **Ruido:**
 - Se añade un pequeño ruido en algunos puntos de la red durante el entrenamiento
 - **Dropout:**
 - Se desactivan algunas unidades durante el entrenamiento para que el resto se adapte en su ausencia.