Ciencia y Tecnología
Secretaría de Ciencia, Humanidades, Tecnología e Innovación

CICESE

Laboratorio de Ciencia de Datos y Aprendizaje Automatico

Curso IA desde Cero

Dr. Irvin Hussein López Nava
M.C. Joan M. Raygoza Romero

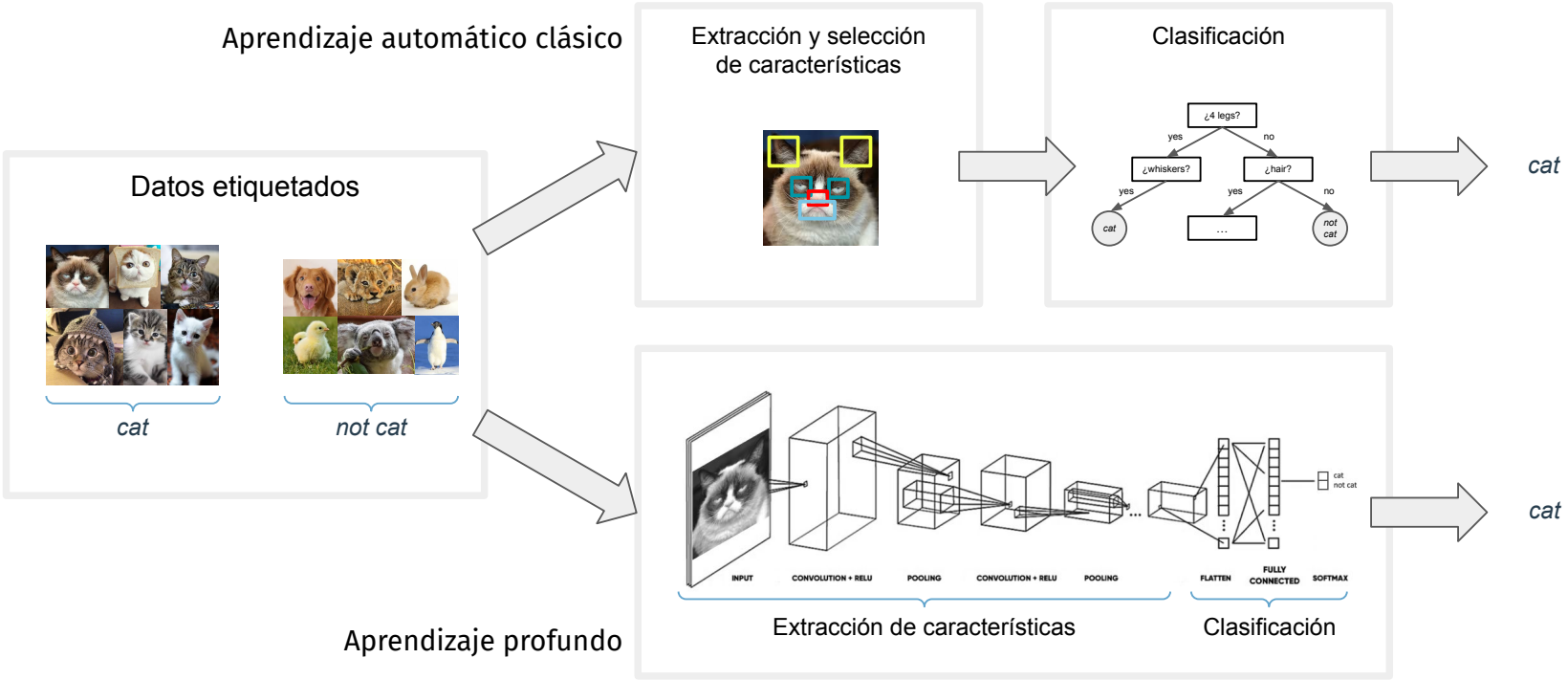
Departamento de Ciencias de la Computación
Centro de Investigación Científica y de Educación Superior de Ensenada
Edición 2025

Un primer caso

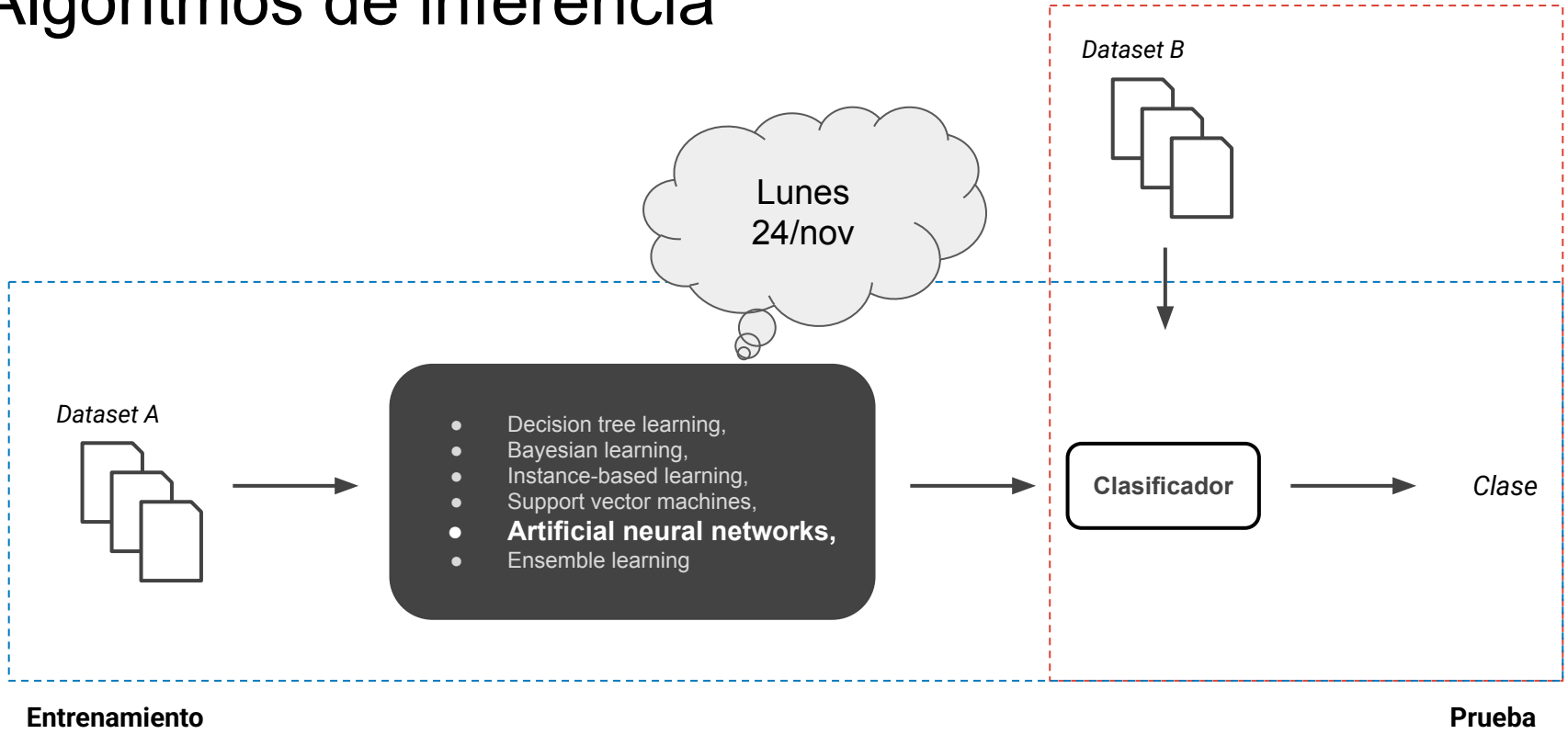
Modelo de clasificación

gato

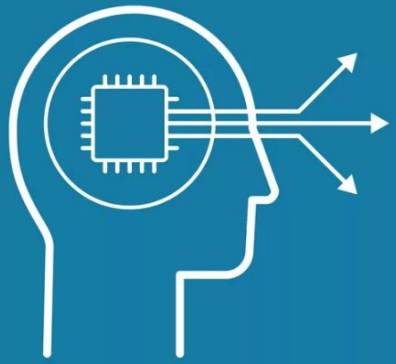
Enfoques de aprendizaje



Algoritmos de inferencia

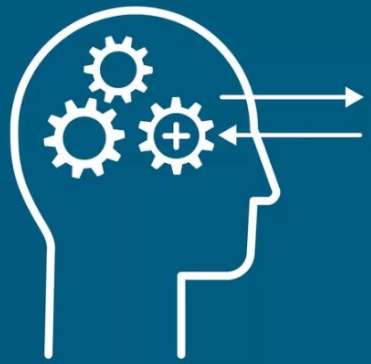


1950 1980 2010



ARTIFICIAL INTELLIGENCE

ENGINEERING OF MACHINES
THAT MIMIC COGNITIVE FUNCTIONS



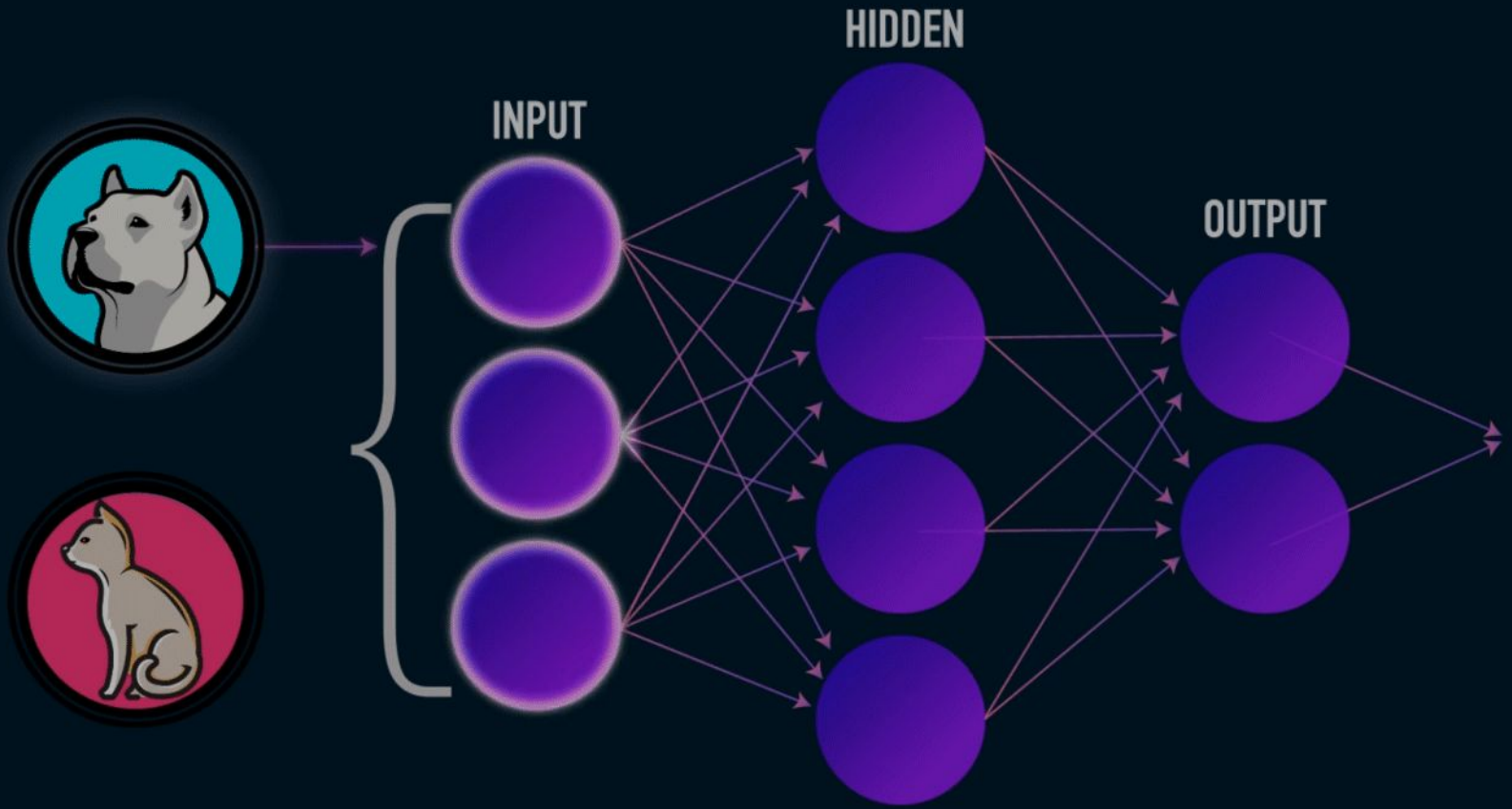
MACHINE LEARNING

ABILITY TO PERFORM TASKS
WITHOUT EXPLICIT INSTRUCTIONS
AND RELYING ON PATTERNS

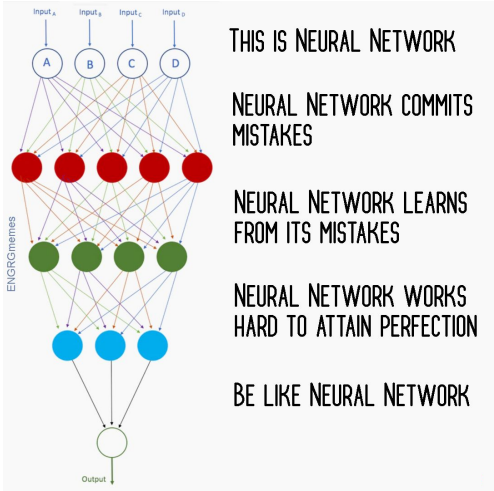
DEEP LEARNING

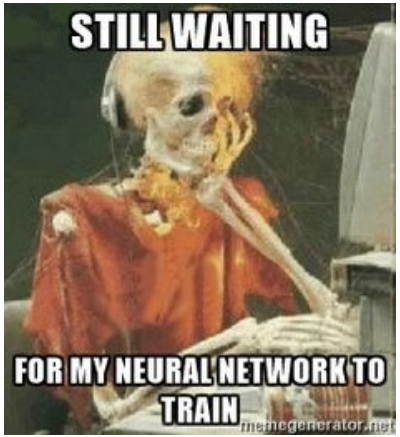
MACHINE LEARNING BASED
ON ARTIFICIAL NEURAL NETWORKS

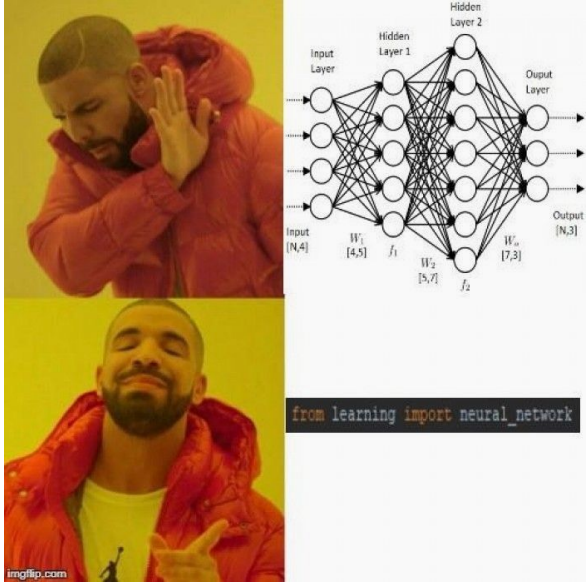
The Beautiful Mind of Neural Nets

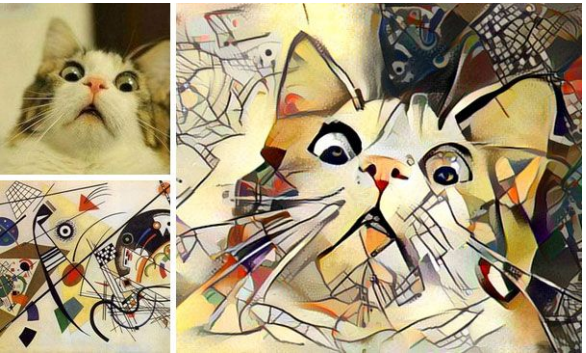


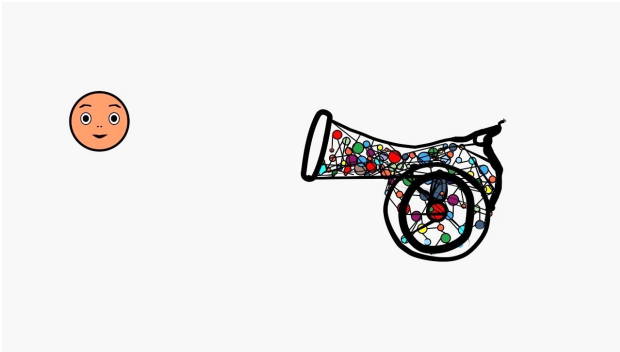
But...


A
B
C
D
Output
ENGRGmemes
THIS IS NEURAL NETWORK
NEURAL NETWORK COMMITS MISTAKES
NEURAL NETWORK LEARNS FROM ITS MISTAKES
NEURAL NETWORK WORKS HARD TO ATTAIN PERFECTION
BE LIKE NEURAL NETWORK

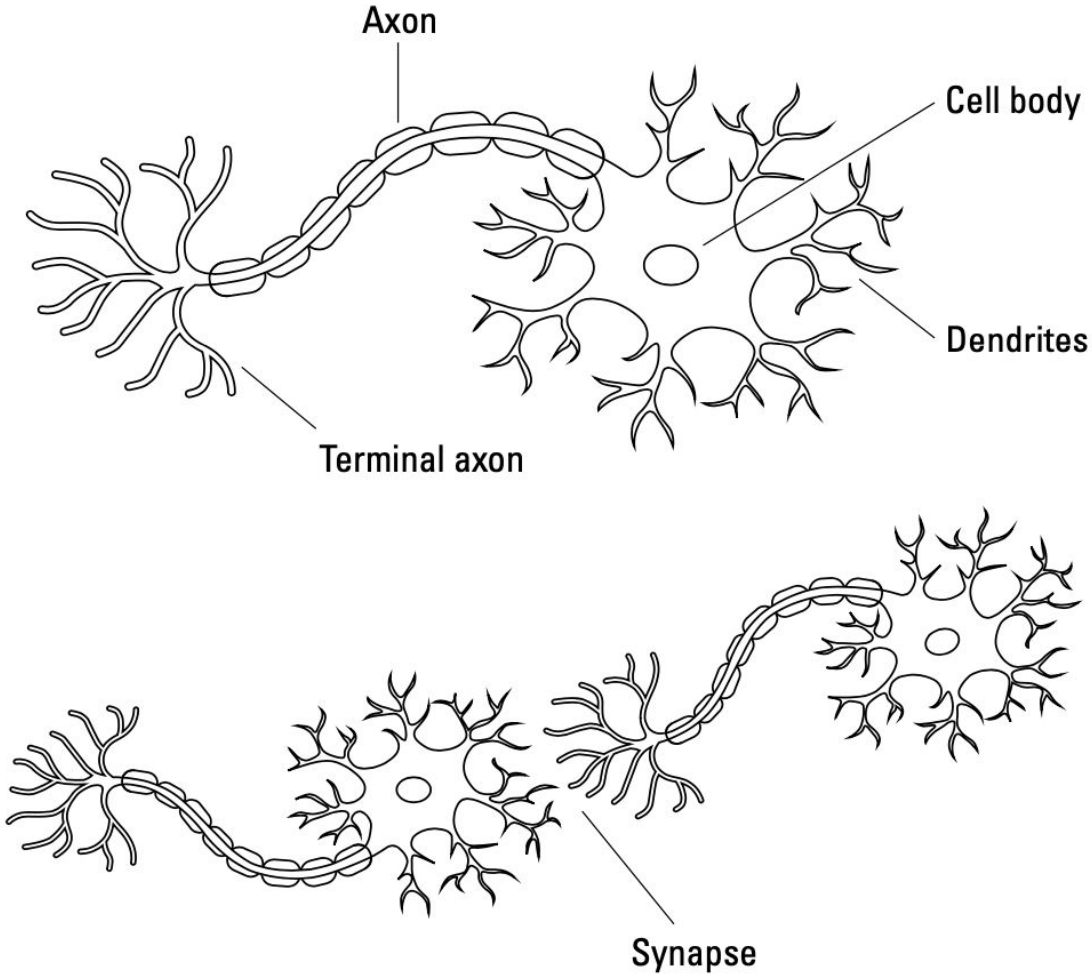

STILL WAITING
FOR MY NEURAL NETWORK TO TRAIN
memegenerator.net


Input Layer
Hidden Layer 1
Hidden Layer 2
Ouput Layer
Input [N,4]
Output [N,3]
from learning import neural_network
imgflip.com

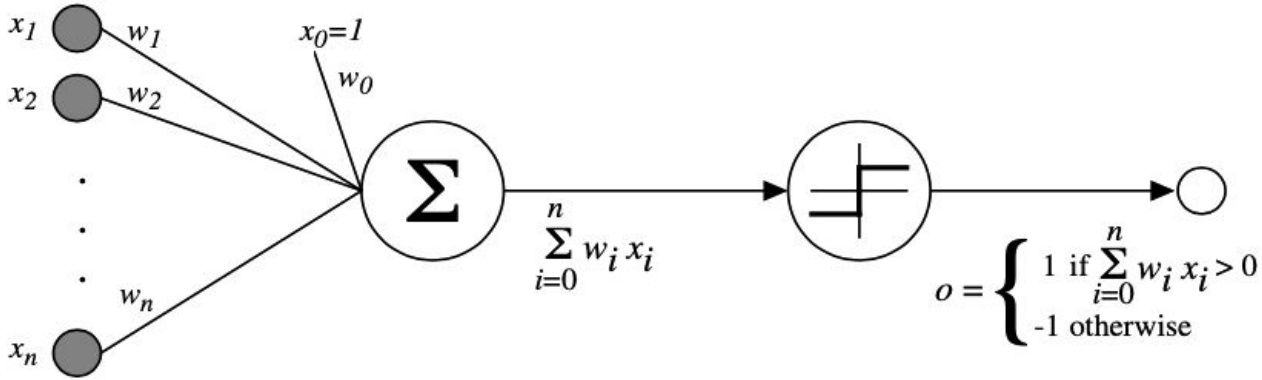




La base


Axon
Cell body
Dendrites
Terminal axon
Synapse

Perceptron

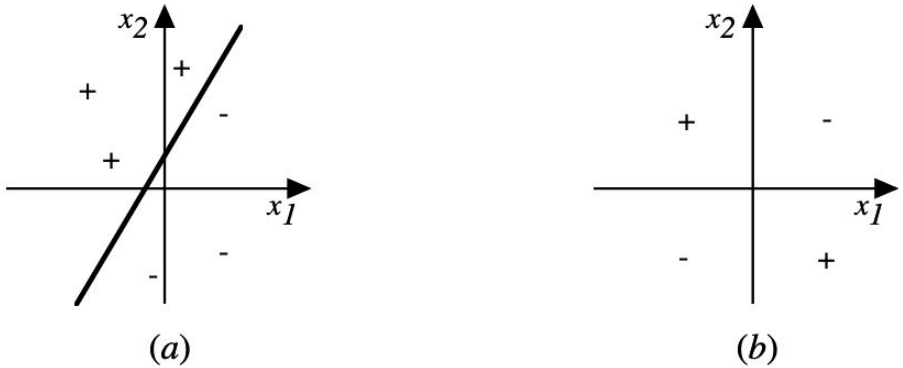


$$o(x_1, ..., x_n) = \begin{cases} 1 \text{ if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 \text{ if otherwise} \end{cases}$$

- O con una notación vectorial:

$$o(\vec{x}) = \begin{cases} 1 \text{ if } \vec{w} \cdot \vec{x} > 0 \\ -1 \text{ if otherwise} \end{cases}$$

Superficie de decisión de un perceptrón



- Representan algunas funciones útiles.
- Aunque algunas funciones no son representables:
- Por ejemplo, no son linealmente separables.
- Por lo tanto, se necesitan ‘redes’.

Regla de entrenamiento del perceptrón

$$w_i \leftarrow w_i + \Delta w_i$$

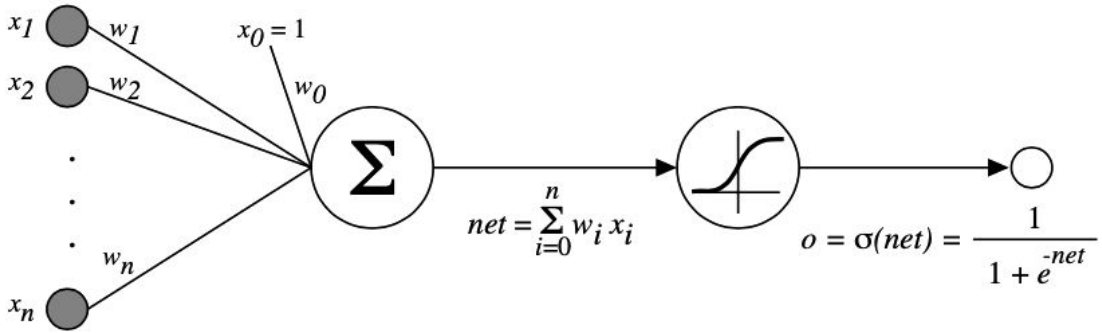
donde

$$\Delta w_i = \eta(t - o)x_i$$

y

- $t = c(\vec{x})$ es el valor objetivo.
- *o* es la salida del perceptrón.
- **η** es una constante pequeña (e.g., 1) llamada tasa de aprendizaje.
- Se puede probar que convergerá:
  - Si los datos de entrenamiento son linealmente separables y **η** suficientemente pequeña.

Unidad sigmoide

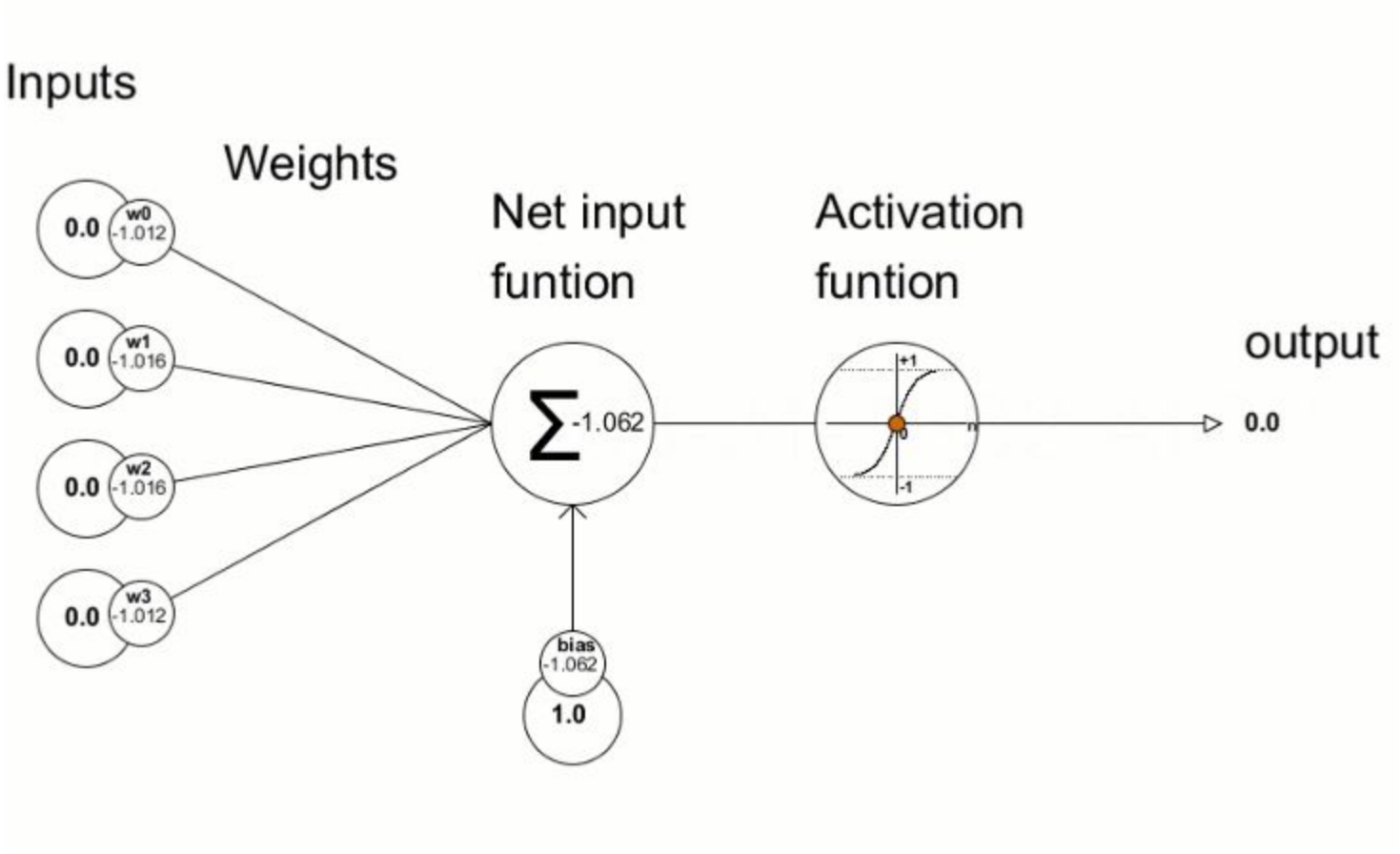


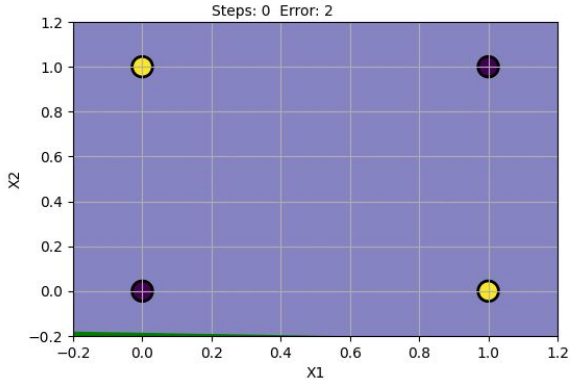
$\sigma(x)$ es la función sigmoidea $\frac{1}{1+e^{-x}}$

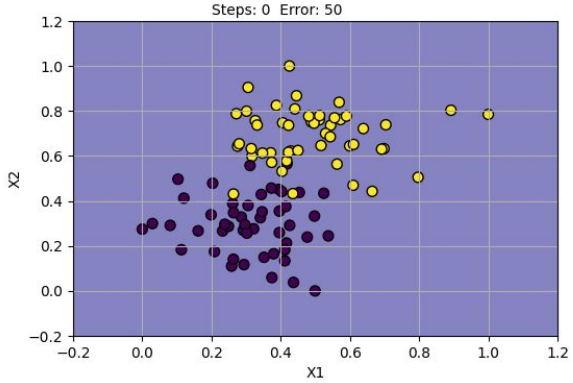
Buena propiedad: $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$

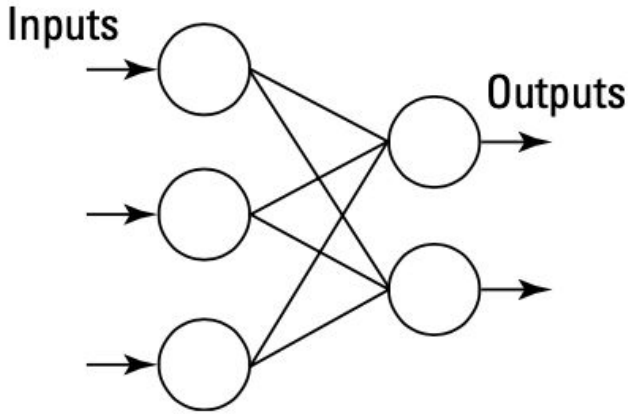
Es posible derivar las reglas de descenso de gradiente para entrenar

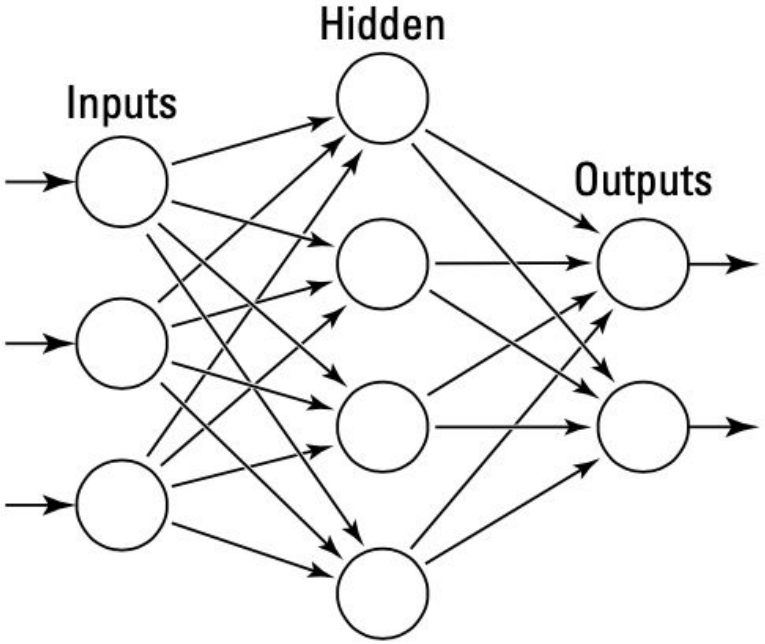
- Una unidad sigmoidea,
- Redes multicapa de unidades sigmoidales → *Backpropagation*


Inputs
Weights
Net input funtion
Activation funtion
output
0.0
w0
-1.012
0.0
w1
-1.016
0.0
w2
-1.016
0.0
w3
-1.012
-1.062
bias
-1.062
1.0
+1
0
-1
0.0


Steps: 0 Error: 2
X1
X2


Steps: 0 Error: 50
X1
X2


Inputs
Outputs


Hidden
Inputs
Outputs

Red Multicapa (MLP)

- Una **red neuronal** consiste en capas de funciones compuestas no lineales, donde cada capa transforma su entrada a través de una combinación lineal más una función de activación.
- Para una capa l, se define:

$$z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)},$$
$$a^{(l)} = \phi(z^{(l)})$$

  donde
  - $W^{(l)}$: matriz de pesos
  - $b^{(l)}$: vector de sesgos
  - ϕ: función de activación (ReLU, sigmoide, etc.)
  - $a^{(0)} = x$: entrada
- Estas transformaciones se aplican capa por capa, generando una salida $\hat{y}$ a partir de una entrada x.

Red Multicapa (MLP)

- Aprender los valores óptimos de $W^{(l)}$, $b^{(l)}$ en todas las capas, minimizando el error entre la salida de la red $\hat{y}$ y la etiqueta real y.
- ¿Cómo se mide ese error? Se usa una función de pérdida $L(\hat{y}, y)$, como:
  - Entropía cruzada para clasificación
  - Error cuadrático medio (MSE) para regresión
- Aquí es donde entra el **descenso de gradiente**: $\theta \leftarrow \theta - \eta \cdot \nabla_\theta L$
  - θ: conjunto de todos los pesos y sesgos en todas las capas
  - η: tasa de aprendizaje
  - $\nabla_\theta L$: gradiente de la pérdida respecto a los parámetros
- Problema: en una red profunda, θ aparece en múltiples capas conectadas, lo que hace necesario aplicar la regla de la cadena para propagar el gradiente desde la salida hacia la entrada.

Red Multicapa (MLP)

- El algoritmo ***backpropagation*** permite calcular $\nabla_\theta L$ de forma eficiente.
- Se basa en dos pasos:
  - **Forward pass**: calcular las activaciones desde la entrada hasta la salida.
  - **Backward pass**: calcular el gradiente de la pérdida respecto a cada parámetro usando la regla de la cadena.
- Ejemplo para una red de 2 capas:

$$
\begin{aligned}
\text{(Output layer)} \quad & \delta^{(L)} = \nabla_{a^{(L)}} \mathcal{L} \odot \phi'(z^{(L)}) \\
\text{(Weight gradient)} \quad & \nabla_{W^{(l)}} \mathcal{L} = \delta^{(l)} (a^{(l-1)})^\top \\
\text{(Hidden layers)} \quad & \delta^{(l)} = (W^{(l+1)\top} \delta^{(l+1)}) \odot \phi'(z^{(l)})
\end{aligned}
$$

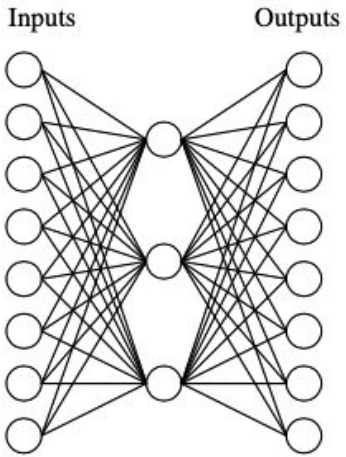
donde $\delta^{(l)}$ representa el error propagado hacia atrás en la capa l.

Red Multicapa (MLP)

- Después de entrenar una **red neuronal** con *backpropagation*, se espera que aprenda una función de mapeo $f_\theta(x) \approx y$ lo suficientemente buena para generalizar a nuevos datos.
    - Pero cuando el modelo se entrena demasiado tiempo o es demasiado complejo, puede terminar ajustándose demasiado al conjunto de entrenamiento, incluyendo ruido y patrones no representativos.
- Este **sobreajuste** se produce cuando el modelo aprende específicamente los datos de entrenamiento pero falla en generalizar a nuevos datos. Indicadores típicos:
    - Pérdida de entrenamiento sigue disminuyendo
    - Pérdida de validación comienza a aumentar
    - Gran discrepancia entre exactitud en entrenamiento vs. validación/test

Aprendizaje de las capas ocultas

A network:



Learned hidden layer representation:

Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.01	.11	.88	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.22	.99	.99	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

Sobreajuste

- ¿Por qué ocurre el sobreajuste? El modelo tiene demasiados parámetros comparado con la cantidad de datos.
  - Se entrena demasiado tiempo sin validación adecuada.
  - El modelo es demasiado expresivo (profundidad, no linealidades, sin restricciones).
  - Los datos tienen ruido, *outliers* o patrones no representativos.
- Diagnóstico del sobreajuste. Se debe dividir el dataset en:
  - ***Training set***: para ajustar pesos
  - ***Validation set:*** para ajustar hiperparámetros y controlar el sobreajuste
  - ***Test set:*** para evaluación final
- Posteriormente, se analizan:
  - Curvas de entrenamiento y validación (*loss* y *accuracy*)
  - Métricas como F1-score, AUC, etc.

Técnicas para reducir el sobreajuste

Regularización L_2 (Ridge)

- Agrega un término de penalización a la función de pérdida:

$$L_{\text{total}} = L_{\text{original}} + \lambda \sum_i w_i^2$$

- Reduce la magnitud de los pesos, haciendo el modelo menos sensible a pequeñas fluctuaciones en los datos.

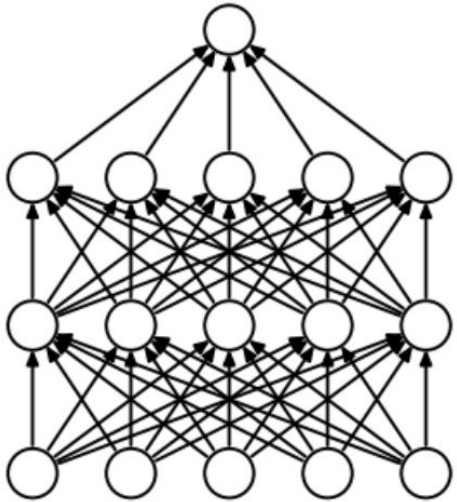
Técnicas para reducir el sobreajuste

Dropout

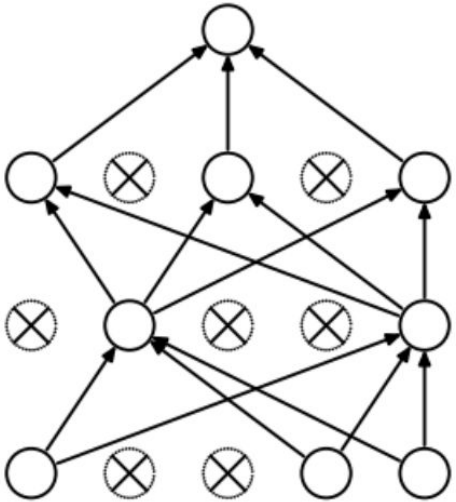
- Durante el entrenamiento, se apagan aleatoriamente algunas neuronas de la red con probabilidad p, lo que evita dependencia excesiva en rutas específicas.

$$a_i' = \begin{cases} 0 & \text{con probabilidad } p \\ \frac{a_i}{1-p} & \text{con probabilidad } 1-p \end{cases}$$

- Implementación:
  - p = *0.2 - 0.5*
  - Solo se usa durante entrenamiento, no en inferencia



(a) Standard Neural Net



(b) After applying dropout.

Técnicas para reducir el sobreajuste

Early Stopping

- Detiene el entrenamiento cuando la pérdida de validación deja de mejorar tras un número fijo de épocas. Previene que el modelo sobreajuste si se entrena demasiado tiempo.

epoch	train_loss	valid_loss	error_rate	time
0	2.300439	1.914173	0.603733	08:21
1	2.001451	1.752479	0.564267	07:59
2	1.847679	1.650396	0.536000	08:00
3	1.657176	1.540547	0.501333	08:00
4	1.529194	1.485027	0.485067	08:00
5	1.435923	1.480115	0.487200	07:59

```
Better model found at epoch 0 with error_rate value: 0.6037333607673645.
Better model found at epoch 1 with error_rate value: 0.5642666816711426.
Better model found at epoch 2 with error_rate value: 0.5360000133514404.
Better model found at epoch 3 with error_rate value: 0.5013333559036255.
Better model found at epoch 4 with error_rate value: 0.48506665229797363.
```

Técnicas para reducir el sobreajuste

Data Augmentation

- Se generan nuevas muestras sintéticas aplicando transformaciones sobre los datos existentes:
  - Rotación, escalado, corte, inversión, modificación de color
  - En visión, aumenta variabilidad y reduce sobreajuste sin necesidad de más datos reales

Técnicas para reducir el sobreajuste

Normalización y Batch Normalization

- La normalización de entrada y el uso de batch normalization en capas ocultas estabilizan el entrenamiento y permiten usar tasas de aprendizaje más altas.

  BatchNorm: $\hat{x}^{(k)} = \frac{x^{(k)} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$ $y^{(k)} = \gamma \hat{x}^{(k)} + \beta$

Reducción de la capacidad del modelo

- Menos neuronas por capa
- Menos capas
- Restricciones sobre activaciones o pesos

Red Multicapa (MLP)

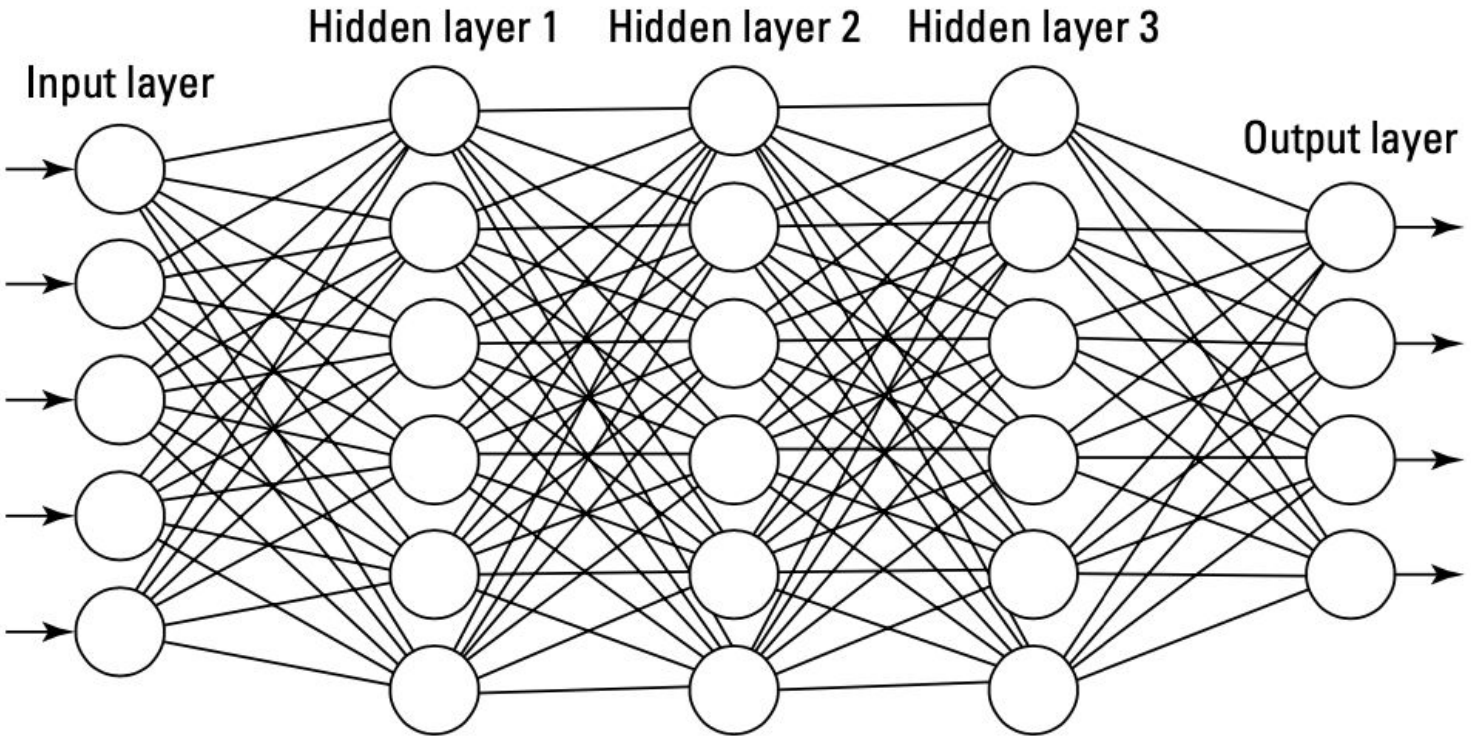
Las MLPs pueden modelar relaciones complejas, pero tienen dos limitaciones clave cuando se aplican directamente a imágenes:

- Número de parámetros inmanejable:
  - Una imagen de entrada 256×256×3 (RGB) tiene 196,608 entradas.
  - Una sola capa densa con 1,000 neuronas requeriría ≈ 197 millones de pesos solo en la primera capa.
- Ignoran estructura espacial:
  - Cada píxel es tratado de forma independiente, perdiendo la noción de vecindad o continuidad.

Solución: diseñar una arquitectura neuronal que: (i) explote estructura local, (ii) reutilice parámetros, (iii) reduzca la cantidad total de pesos.

- Esto da lugar a las Redes Neuronales Convolucionales (CNNs).

Aprendizaje profundo


Input layer
Hidden layer 1
Hidden layer 2
Hidden layer 3
Output layer

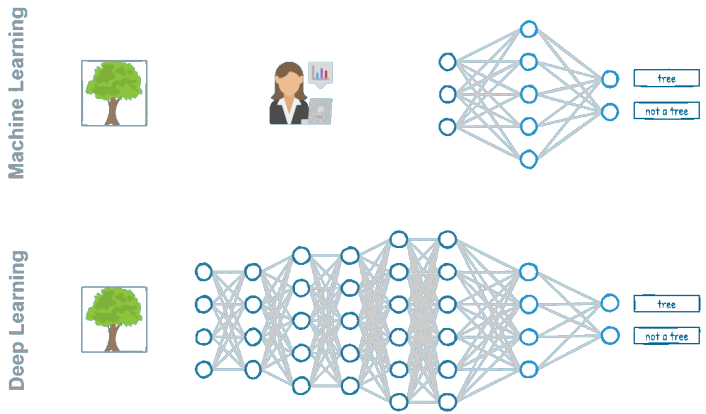
¿Qué es el aprendizaje profundo?

Es un conjunto de algoritmos de **aprendizaje automático** que intentan modelar abstracciones de alto nivel en datos usando arquitecturas compuestas de transformaciones no-lineales.

LeCun, Y., Bengio, Y., & Hinton, G. (2015).
Deep learning. nature, 521(7553), 436-444.

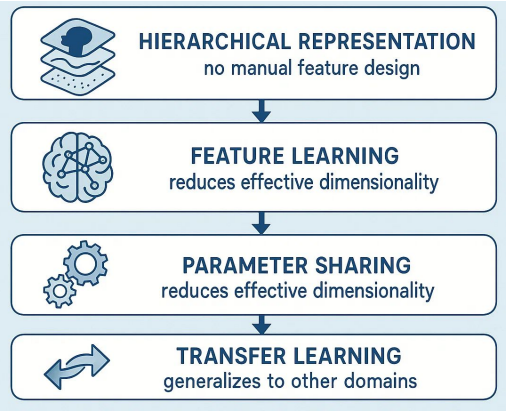
¿Por qué profundo?

- La **profundidad** se refiere a la cantidad de capas de representación que una red tiene.
- Mientras las redes superficiales (una sola capa oculta, e.g. MLP) pueden aproximar funciones arbitrarias, las redes profundas pueden hacerlo con una representación más eficiente y jerárquica.
- Ejemplo clásico: una red profunda puede representar bordes → formas → partes → objetos.



Ventajas clave

- **Representación jerárquica**:
  del nivel más bajo al más semántico.
- **Aprendizaje de características**:
  no requiere diseño manual de features.
- **Compartición de parámetros**:
  reduce dimensionalidad efectiva.
- **Transferencia de conocimiento**:
  aprendizaje generalizable a otros dominios.



Más sobre el *deep learning*

El **aprendizaje profundo** permite que redes neuronales con varios niveles de neuronas **aprendan a representar características**, sin decirles cuáles son.

Estos modelos utilizan combinaciones de aprendizaje supervisado y no supervisado, en los diferentes niveles.

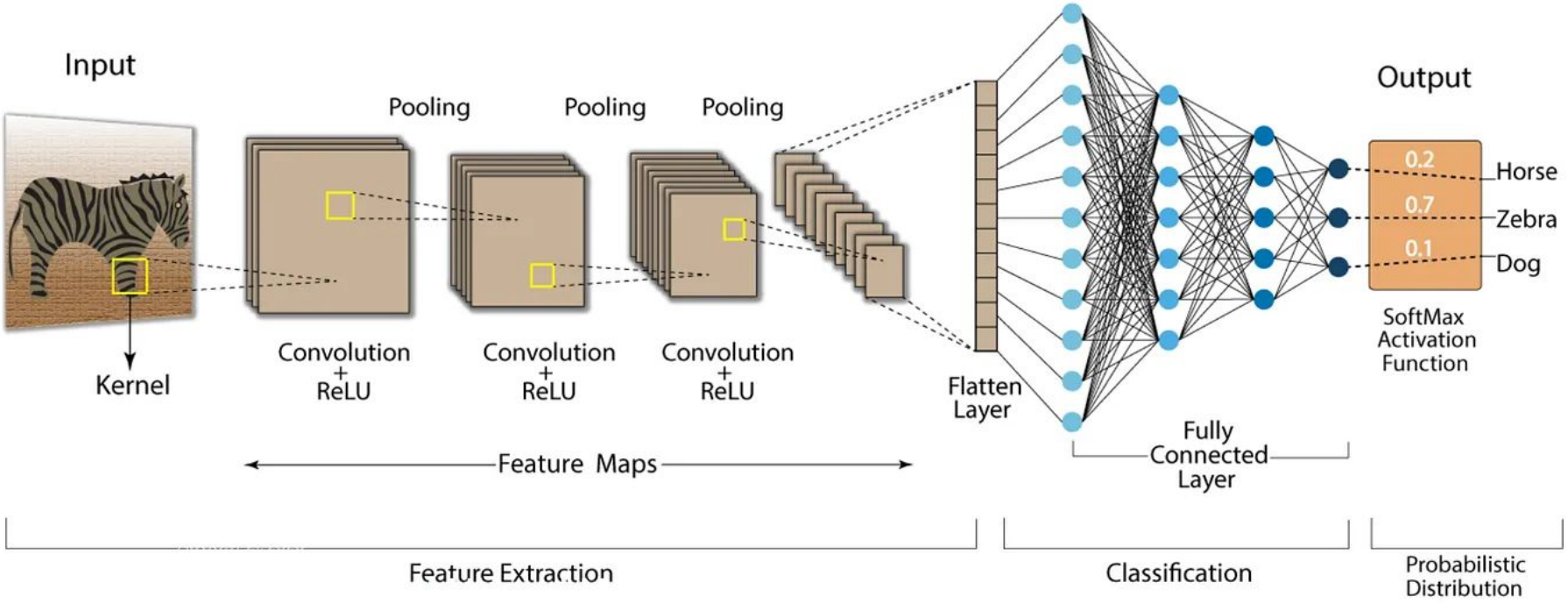
Las arquitecturas más populares son:

- Red profunda de convolución (CNN)
- Red de creencias profundas (deep-belief network)
- Redes recurrentes de memoria corta y larga (long-short term memory, LSTM)
- Transformers
- Redes Generativas adversarias (GAN)

Redes Neuronales Convolucionales CNNs

Las CNN fueron diseñadas con tres ideas clave:

- **Conectividad local**: en lugar de conectar cada neurona a toda la entrada, se conecta solo a una región pequeña (*receptive field*).
- **Pesos compartidos**: un mismo conjunto de pesos (filtro o *kernel*) se aplica en toda la imagen.
- **Submuestreo**: reduce la resolución para controlar complejidad y ganar invariancia.


Input
Pooling
Pooling
Pooling
Output
0.2
Horse
0.7
Zebra
0.1
Dog
Kernel
Convolution + ReLU
Convolution + ReLU
Convolution + ReLU
Flatten Layer
SoftMax Activation Function
Feature Maps
Fully Connected Layer
Feature Extraction
Classification
Probabilistic Distribution

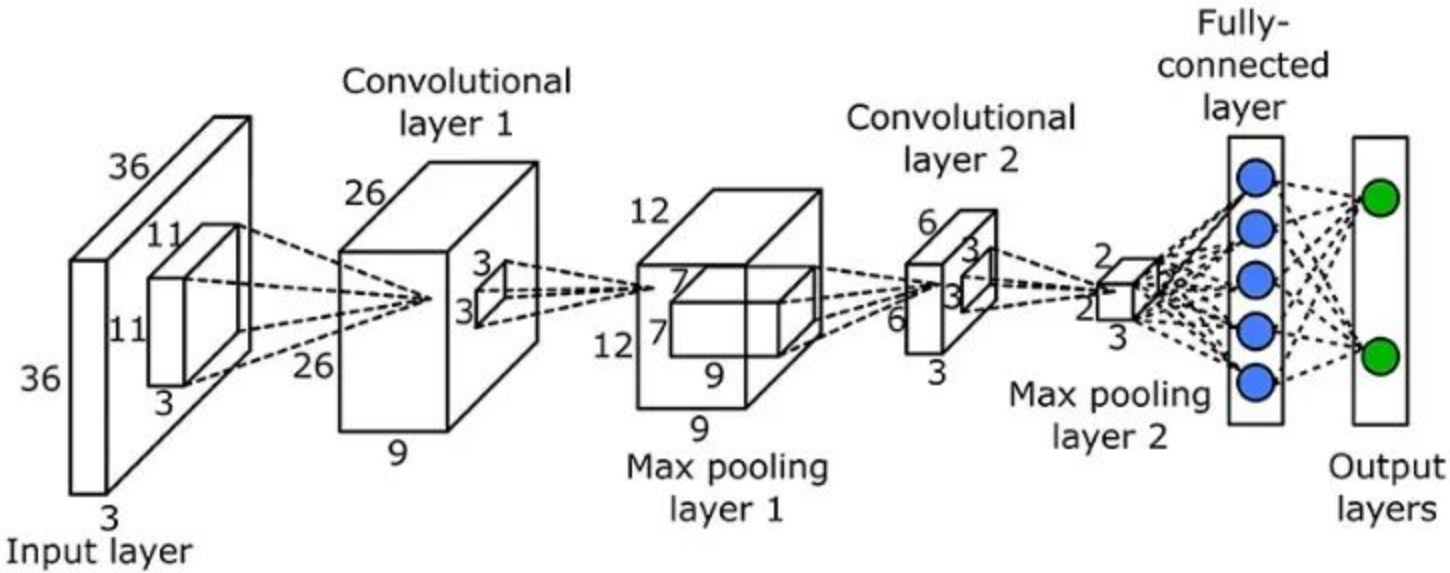
Capa convolucional

- **Operación principal:** convolución discreta

- Cada filtro $w \in R^{k \times k}$ se desliza sobre la imagen y produce una activación para cada posición:

$$y_{i,j} = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} w_{m,n} \cdot x_{i+m,j+n}$$

- Se aplican múltiples filtros F, generando varios mapas de activación.
- *Output*: tensor 3D (altura × anchura × filtros)


36
11
11
36
3
3
Input layer
Convolutional
layer 1
26
3
3
26
9
12
7
7
12
9
9
Max pooling
layer 1
Convolutional
layer 2
6
3
3
6
3
Max pooling
layer 2
Fully-
connected
layer
Output
layers

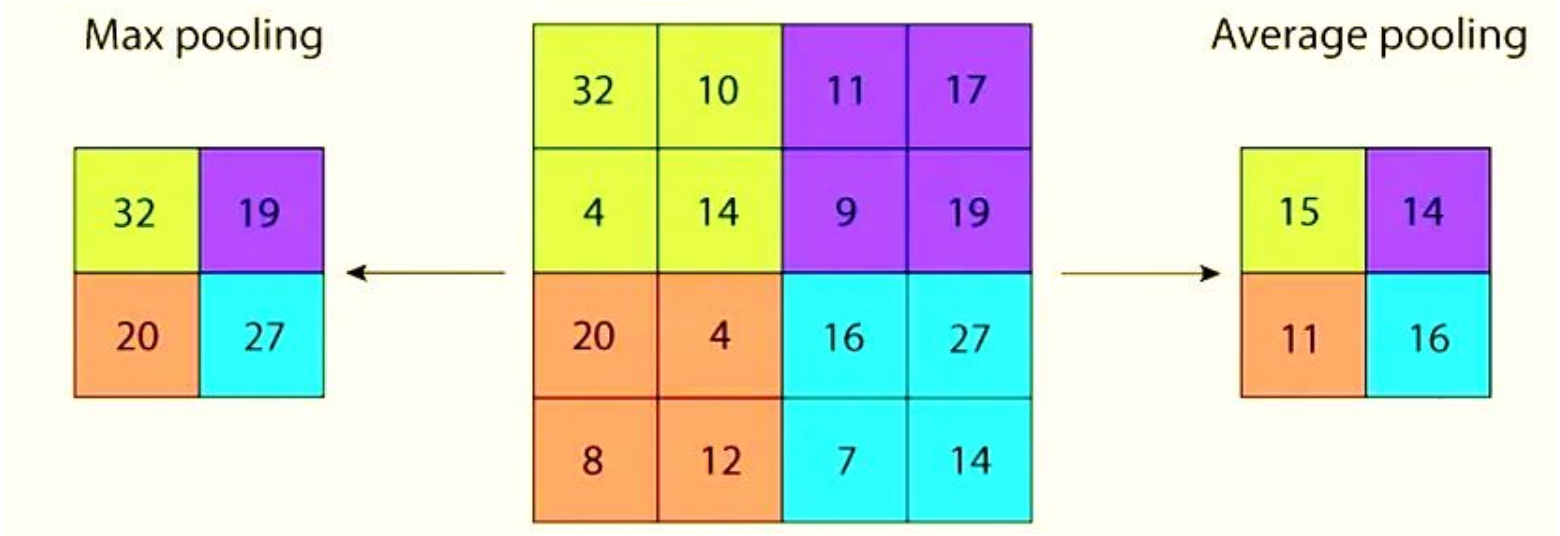
Pooling (submuestreo)

- **Objetivo:** reducir la dimensión espacial y aumentar invariancia.

- *Max pooling (2×2)*: toma el máximo valor en una ventana
- *Average pooling:* toma el promedio

- Ejemplo:

$$\text{MaxPool}\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix} = 4$$

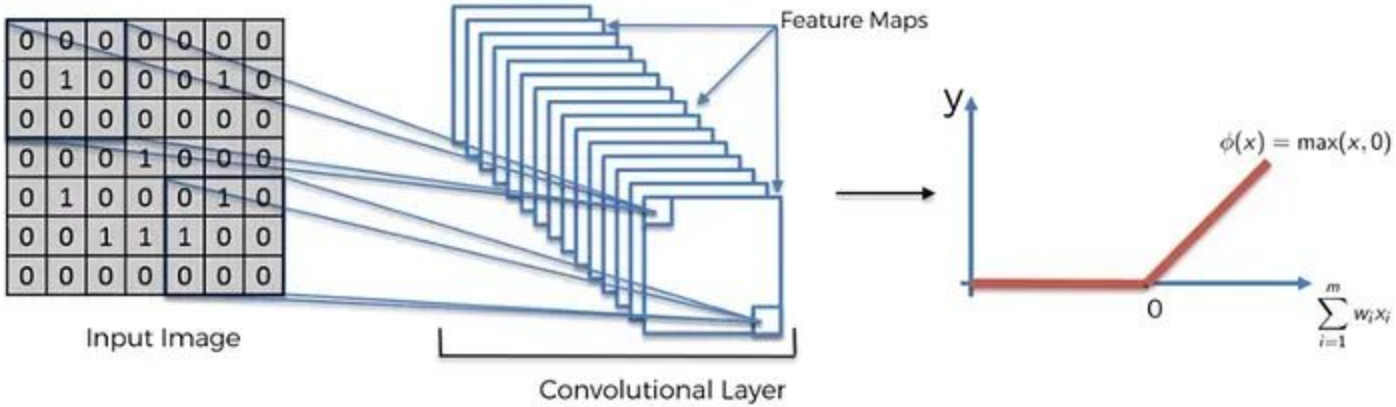

Max pooling
32
19
20
27
32
10
11
17
4
14
9
19
20
4
16
27
8
12
7
14
Average pooling
15
14
11
16

Activaciones no lineales

- Después de cada convolución se aplica una función de activación no lineal (usualmente ReLU):

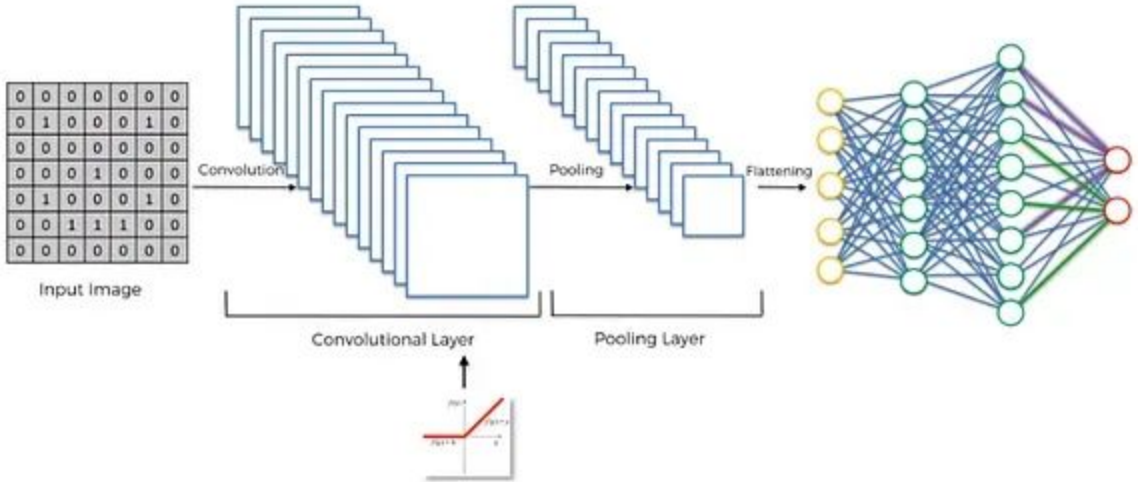
$$ReLU(x) = max(0, x)$$

- Esto introduce no linealidades necesarias para aproximar funciones complejas, como ya se explicó en MLP.


0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 1 0 0 0 1 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
Input Image
Feature Maps
Convolutional Layer
y
$\phi(x) = \max(x, 0)$
0
$\sum_{i=1}^{m} w_i x_i$

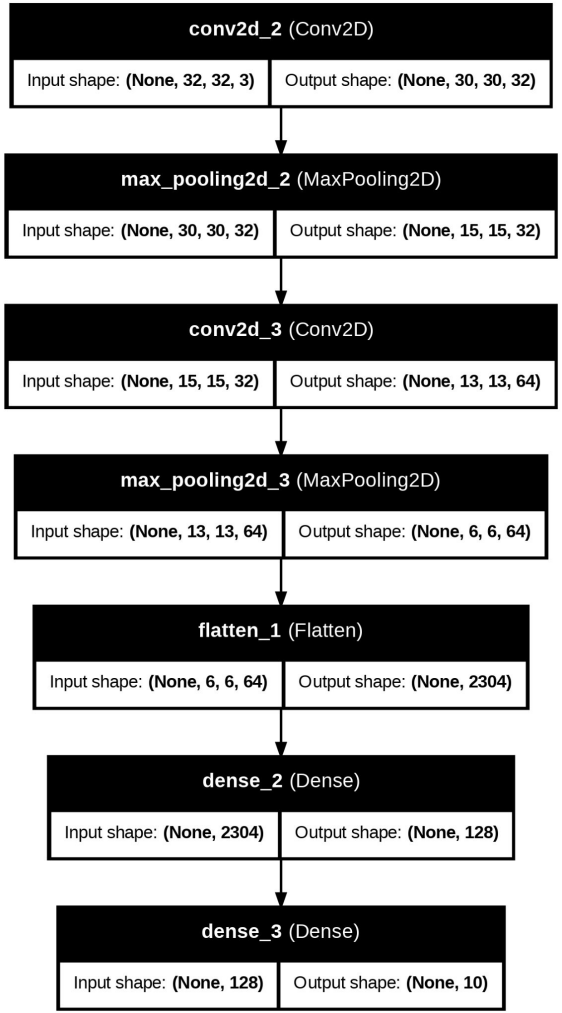
Capas completamente conectadas

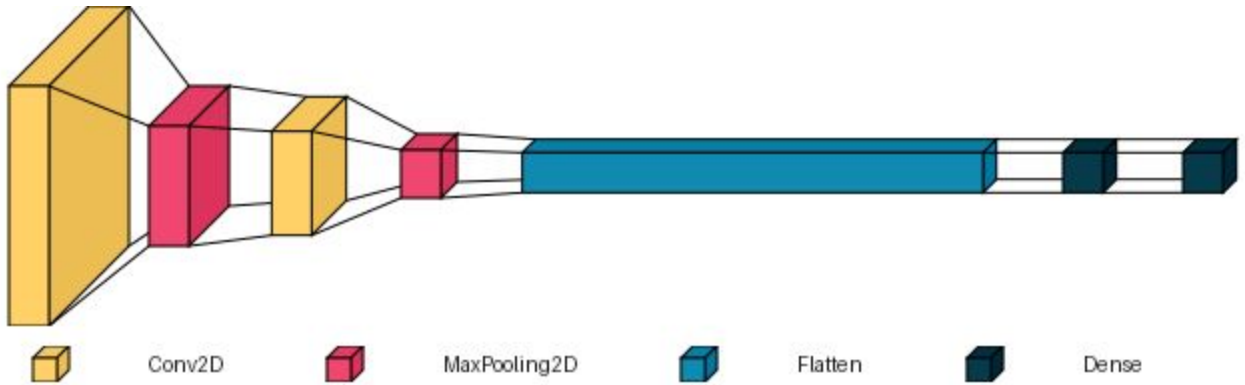
- En una capa densa (*dense layer*) cada unidad está conectada con todas las unidades de la capa anterior, exactamente como en un MLP.
  - En una CNN, las capas FC suelen estar al final de la red, después de la extracción de características convolucionales y del aplanamiento (flatten) del mapa de activaciones.
- Las capas FC tienen dos propósitos principales:
  - Combinar las características extraídas por las capas convolucionales y de pooling para producir una decisión global.
  - Actuar como un clasificador no lineal que mapea los vectores de características a etiquetas de clase (en clasificación) o a predicciones continuas (en regresión).
- Para conectar las salidas de una CNN a una FC, primero se deben convertir los tensores de salida 3D (e.g., 7×7×512) a un vector 1D = 25,088 elementos ⇒ vector de entrada para FC


0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 1 0 0 0 1 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
Input Image
Convolution
Pooling
Flattening
Convolutional Layer
Pooling Layer

Arquitectura típica de una CNN

Etapa	Operación	Output shape (ejemplo)
Entrada	Imagen RGB 32×32×3	32×32×3
Conv (3×3, 32)	ReLU	30×30×3
MaxPool (2×2)	↓	15×15×3
Conv (3×3, 64)	ReLU	13×13×64
MaxPool (2×2)	↓	6×6×64
Flatten	—	2304
FC (128)	ReLU	128
FC (10)	Softmax	n clases, e.g. 10


conv2d_2 (Conv2D)
Input shape: (None, 32, 32, 3)
Output shape: (None, 30, 30, 32)
max_pooling2d_2 (MaxPooling2D)
Input shape: (None, 30, 30, 32)
Output shape: (None, 15, 15, 32)
conv2d_3 (Conv2D)
Input shape: (None, 15, 15, 32)
Output shape: (None, 13, 13, 64)
max_pooling2d_3 (MaxPooling2D)
Input shape: (None, 13, 13, 64)
Output shape: (None, 6, 6, 64)
flatten_1 (Flatten)
Input shape: (None, 6, 6, 64)
Output shape: (None, 2304)
dense_2 (Dense)
Input shape: (None, 2304)
Output shape: (None, 128)
dense_3 (Dense)
Input shape: (None, 128)
Output shape: (None, 10)


Conv2D
MaxPooling2D
Flatten
Dense

Entrenamiento de las CNNs

Las CNNs se entrenan de la misma forma que los MLPs:

- ***Forward pass:*** se propagan las entradas a través de convoluciones, activaciones y capas densas.
- ***Loss:*** se calcula con entropía cruzada o MSE.
- ***Backward pass:*** se aplica backpropagation para actualizar:
  - Pesos de filtros convolucionales
  - Pesos de capas densas
- ***Optimizador:*** descenso de gradiente, Adam, RMSprop, etc.

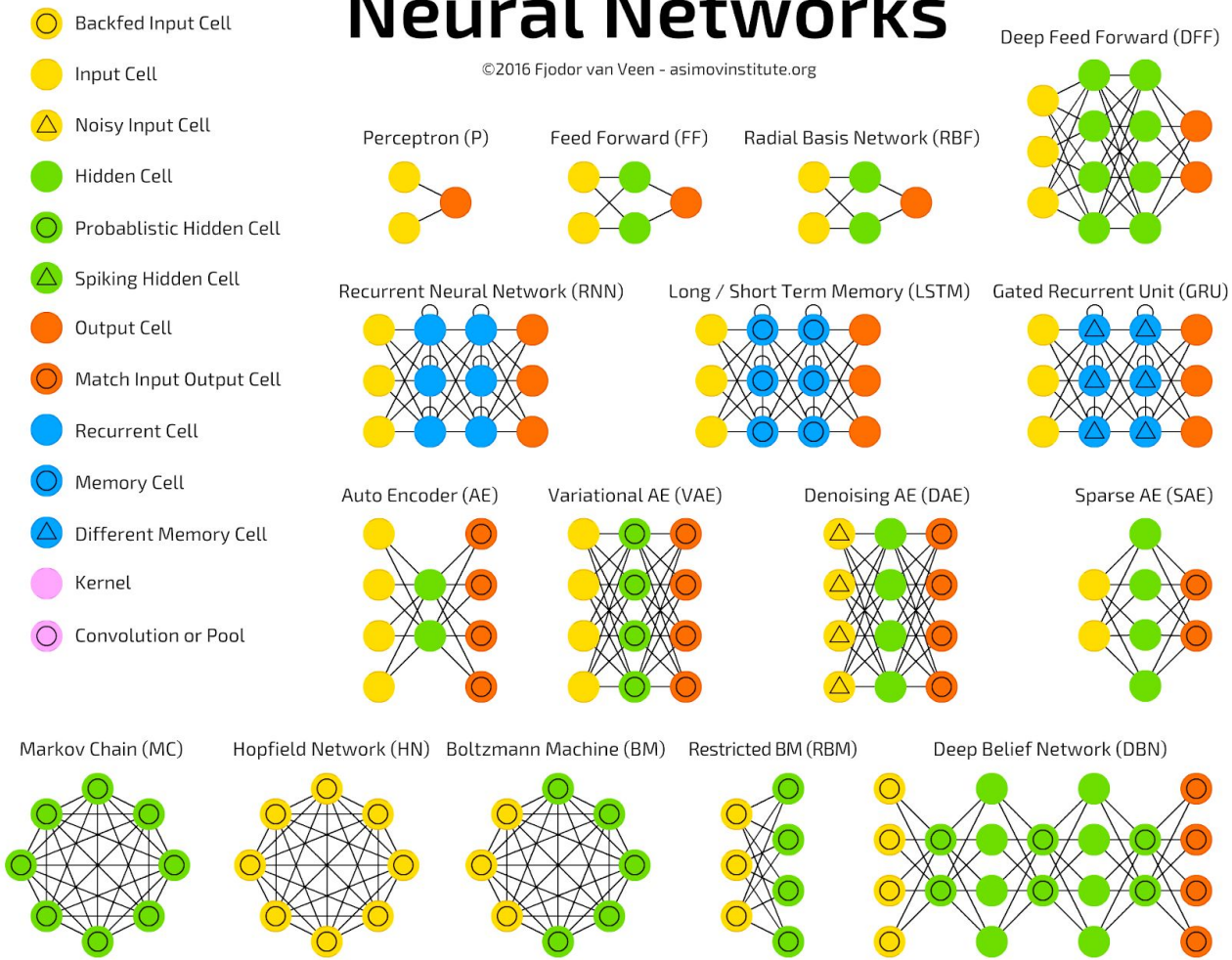
Todas las técnicas para evitar el sobreajuste vistas antes también aplican a CNNs: *Dropout, Regularización L^2, Early stopping, Data augmentation*

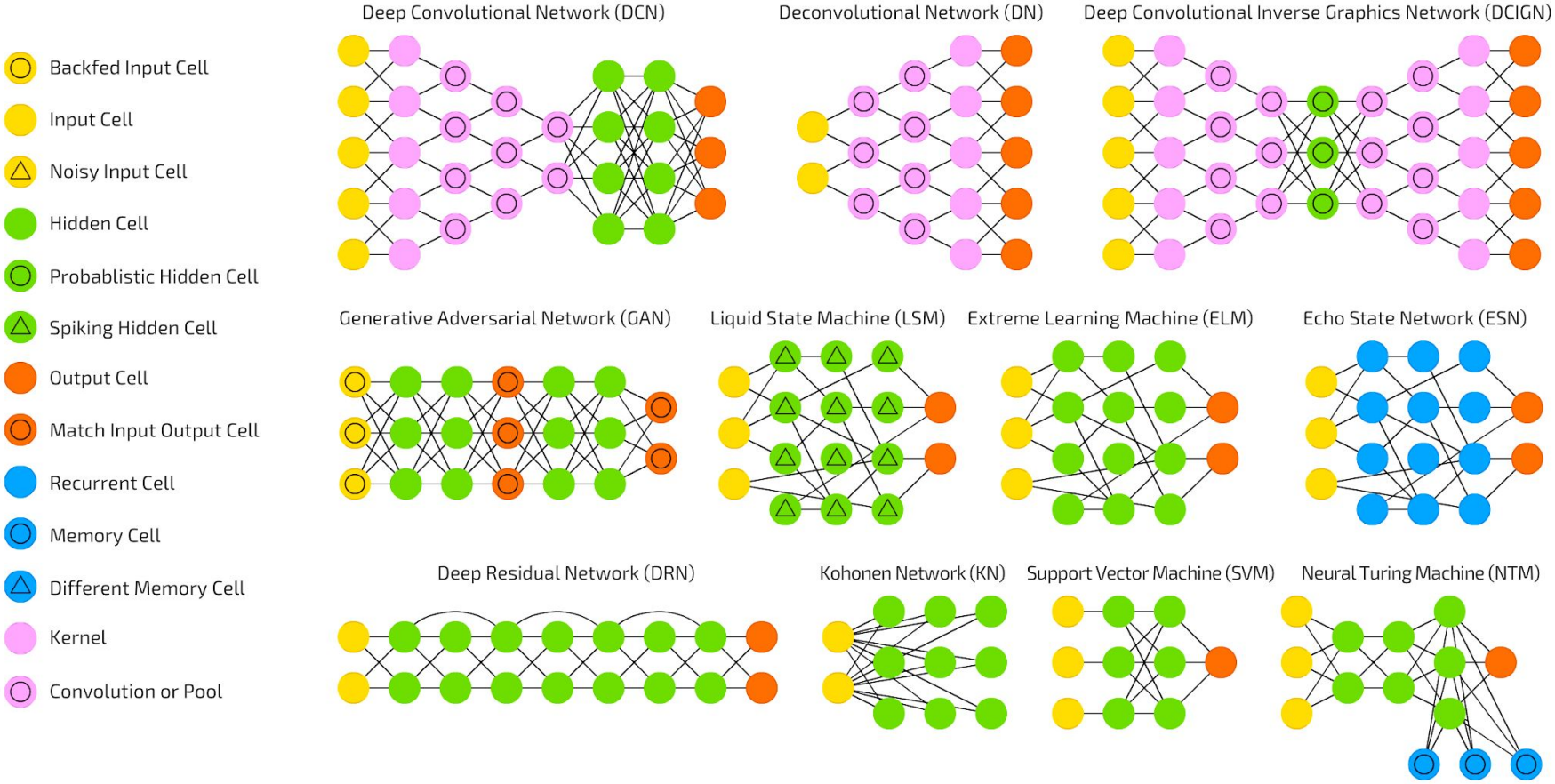
Aplicaciones de las CNNs

- Clasificación de imágenes (ImageNet)
- Detección de objetos (YOLO, Faster R-CNN)
- Segmentación semántica (U-Net, DeepLab)
- Reconocimiento facial (FaceNet, ArcFace)
- Visión médica (clasificación, segmentación)
- Series temporales (1D-CNN)
- Audio, texto embebido en imagen

A mostly complete chart of

Neural Networks




Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

¿Preguntas?

hussein@cicese.mx

SCAN ME