

# Aprendizaje Profundo

Maestría en Modelación y Optimización de Procesos  
CIMAT-Aguascalientes

Dra. Lilí Guadarrama Bustos<sup>1</sup>

Dr. Isidro Gómez-Vargas<sup>2</sup>

<sup>1</sup>Cátedra CONACyT CIMAT-Aguascalientes

<sup>2</sup>Investigador posdoctoral en ICF-UNAM

Clase del semestre enero-julio 2022

# Contenido

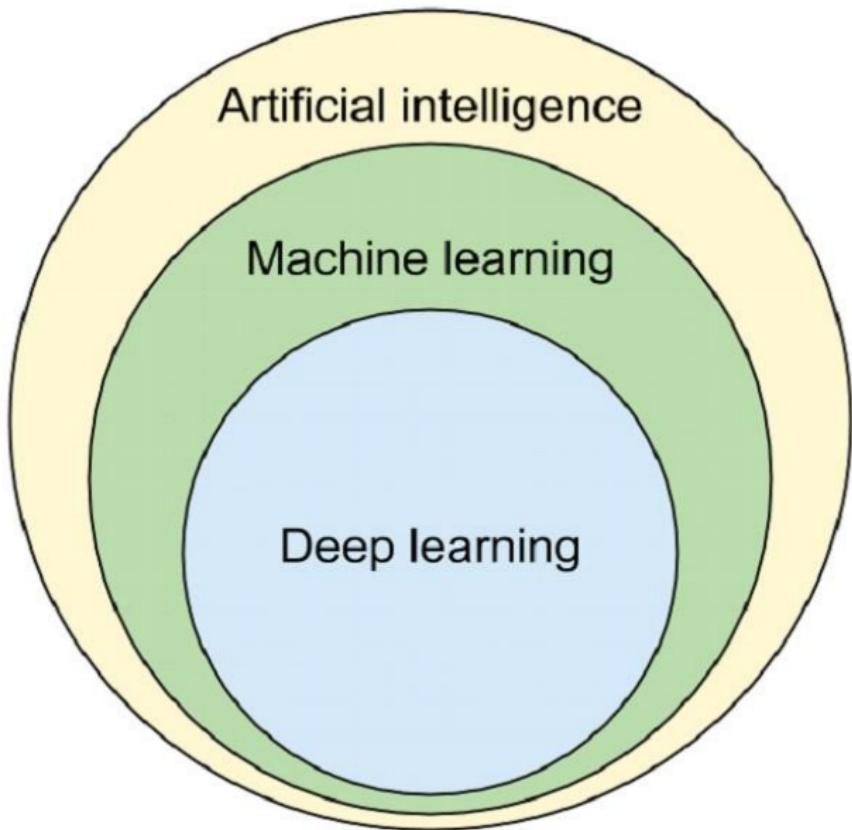
## 1 Clase 1: Bases de Aprendizaje Automático

- Algoritmos de aprendizaje.
- Capacidad, subajuste y sobreajuste
- Hiperparámetros y conjuntos de validación.
- Estimadores, sesgo y varianza.
- Estimación de Máxima Verosimilitud(MLE).
- Estadística Bayesiana.
- Algoritmos de aprendizaje supervisado.
- Descenso del gradiente estocástico.
- Desafios actuales Aprendizaje Profundo.

## 2 Clase 2: Deep Feedforward Networks

- 6.1 Ejemplo: Aprendiendo XOR
- Gradient-based learning
- Unidades ocultas
- Diseño de arquitectura
- Backpropagation y otros algoritmos de diferenciación
- Regularización

# Aprendizaje profundo



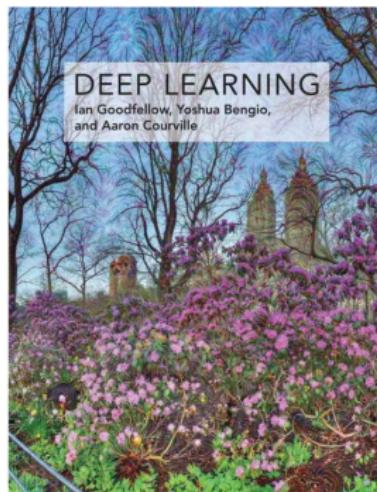
# Bibliografía

## Bibliografía principal

*Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.*

## Diapositivas:

- [https://www.deeplearningbook.org/lecture\\_slides.html](https://www.deeplearningbook.org/lecture_slides.html)
- <https://github.com/InfolabAI/DeepLearning>



<b>Born</b>	March 5, 1964 (age 57) Paris, France
<b>Citizenship</b>	Canada
<b>Alma mater</b>	McGill University
<b>Known for</b>	Deep learning, neural machine translation, generative adversarial networks, "attention model"®, word embeddings, denoising auto-encoders, neural language models, learning to learn Marie-Victorin Prize (2012) Turing Award (2018) AAAI Fellow (2019)
<b>Awards</b>	
<b>Scientific career</b>	



<b>Born</b>	1985/1986 (age 35–36)
<b>Nationality</b>	American
<b>Alma mater</b>	Stanford University Université de Montréal
<b>Known for</b>	Generative adversarial networks, Adversarial Examples
<b>Scientific career</b>	
<b>Fields</b>	Computer science
<b>Institutions</b>	Apple Inc. Google Brain OpenAI
<b>Thesis</b>	Deep Learning of Representations and its Application to Computer Vision (2014)
<b>Doctoral advisor</b>	Yoshua Bengio
<b>Website</b>	<a href="http://www.iangoodfellow.com">www.iangoodfellow.com</a>



Aaron Courville

Université de Montréal

Dirección de correo verificada de umontreal.ca - [Página personal](#)  
Machine learning Artificial Intelligence

### TÍTULO

### CITADO POR

#### Generative adversarial nets

I Goodfellow, J Pouget-Abadie, M Mirza, B Xu, D Warde-Farley, S Ozair, ... Advances in neural information processing systems 27

40148

#### Deep learning

I Goodfellow, Y Bengio, A Courville  
Nature

36121

#### Representation learning: A review and new perspectives

Y Bengio, A Courville, P Vincent  
IEEE transactions on pattern analysis and machine intelligence 35 (8), 1770-1828

10333

# Temario

Se abordarán partes de los siguientes capítulos:

- Capítulo 5: Bases del Aprendizaje Automático.
- Capítulo 6: Redes neuronales profundas de propagación hacia adelante.
- Capítulo 7: Regularización.
- Capítulo 8: Optimización para el entrenamiento de modelos profundos.

# Temario

- Capítulo 9: Redes neuronales convolucionales.
- Capítulo 10: Modelación secuencial (redes recurrentes y recursivas).
- Capítulo 11: Metodología práctica.
- Capítulo 12: Aplicaciones.
- Temas elegidos por el grupo de la parte III del libro (capítulos 13-20).

# Algoritmos de aprendizaje.

“ Un programa de computadora se dice que aprende de la experiencia  $E$  con respecto a cierta clase de tareas  $T$  y con medida de rendimiento  $P$ , si su rendimiento en las tareas en  $T$ , medido por  $P$ , mejora con la experiencia  $E$ . ”

Tom Mitchell, 1997

# Glosario

- **Ejemplo.**  $x \in \mathbf{R}^n$ , colección de  $n$  características.
- **Conjunto de datos.** Colección de ejemplos.
- **Características.** Atributos.
- **Matriz de datos.**

# Tareas T

- Clasificación
- Regresión
- Transcripción
- Traducción
- Salidas estructuradas
- Detección de anomalías
- Síntesis y muestreo
- Imputing valores perdidos
- Quitar ruido
- Estimación de densidad o probabilidad

# La medida de rendimiento P

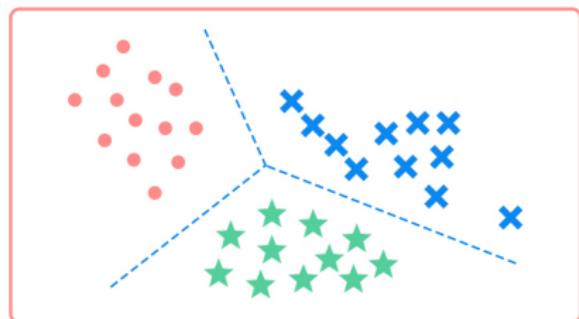
- Exactitud  $\frac{TP+TN}{TP+TN+FP+FN}$
- Error cuadrático medio  $MSE = \frac{1}{n} \sum_i^n ||\hat{y}_i - y_i||$
- Densidad de probabilidad
- etc

# La experiencia, E



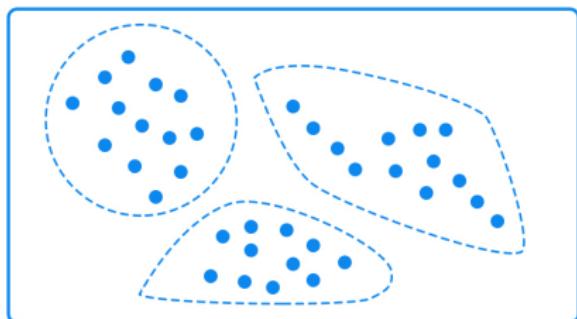
## Supervised vs. Unsupervised Learning

Classification



Supervised learning

Clustering



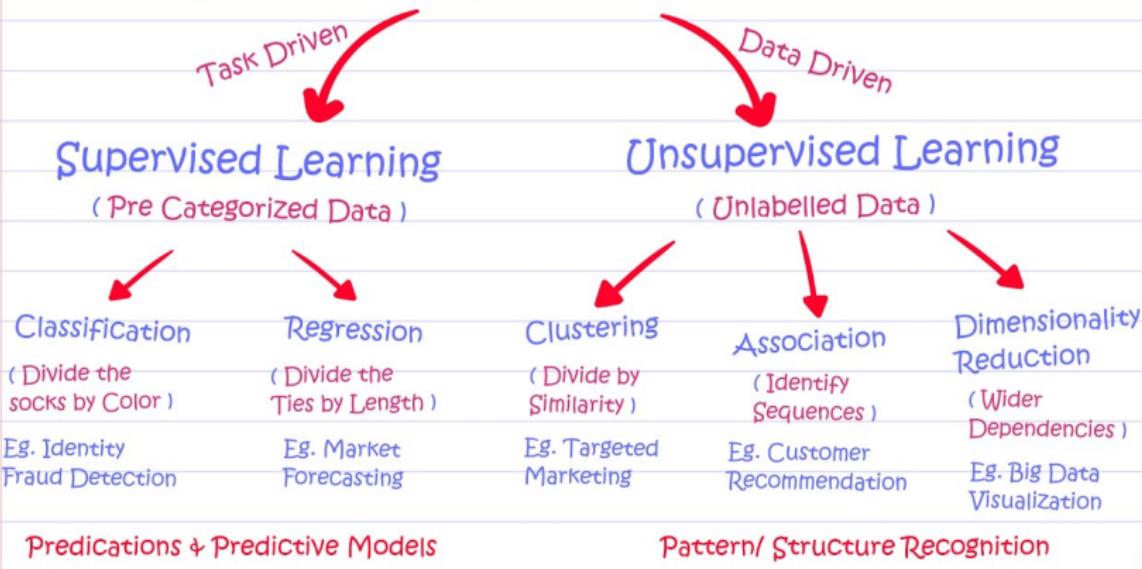
Unsupervised learning

Fuente:

[https://analystprep.com/study-notes/cfa-level-2/quantitative-method/  
supervised-machine-learning-unsupervised-machine-learning-deep-learning/  
attachment/img\\_12-4/](https://analystprep.com/study-notes/cfa-level-2/quantitative-method/supervised-machine-learning-unsupervised-machine-learning-deep-learning/attachment/img_12-4/)

# La experiencia, E

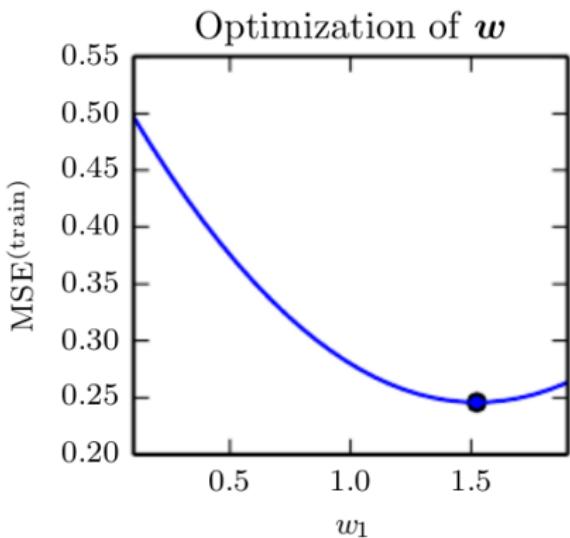
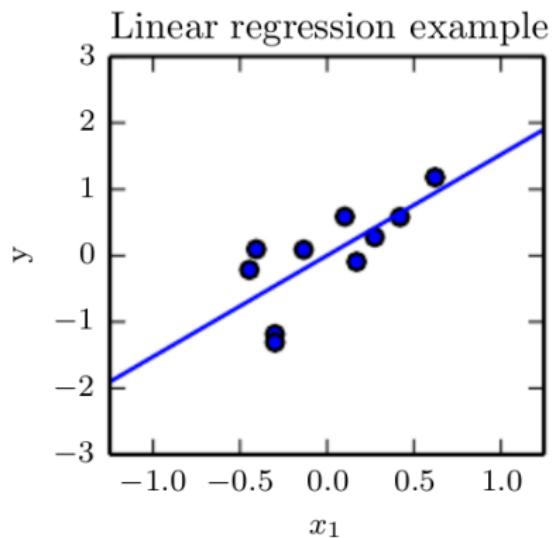
## Classical Machine Learning



Fuente: <https://medium.com/@recrosoft.io/supervised-vs-unsupervised-learning-key-differences-cdd46206cdcb>



## Ejemplo: regresión lineal

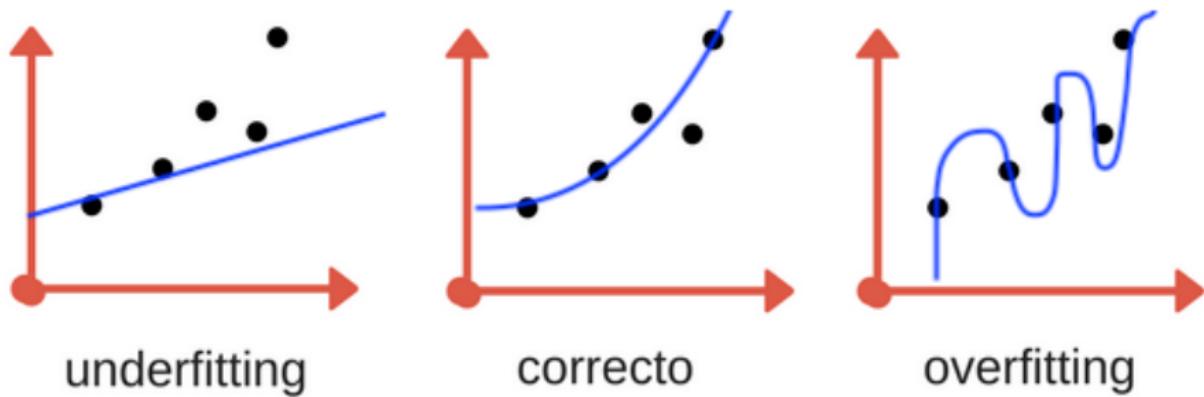


Ver páginas 107-108 del libro.

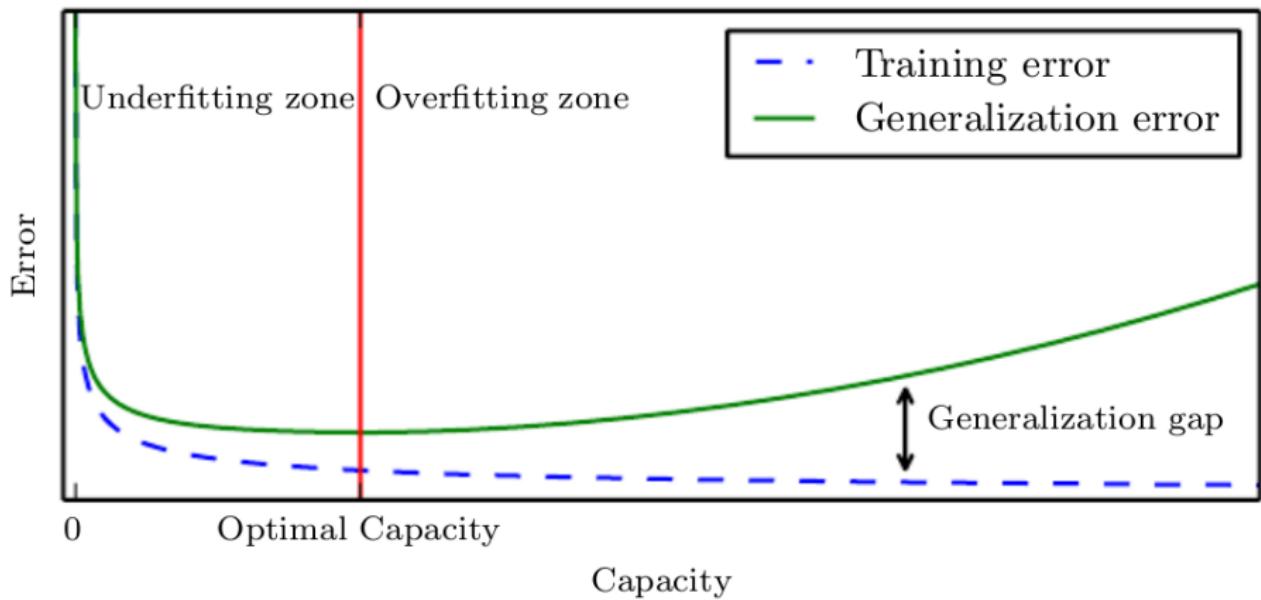
# Capacidad, subajuste y sobreajuste

- Algo que separa al aprendizaje automático de la optimización es el uso de un error de generalización o error de prueba (*test error*).
- Hay que minimizar tanto el error de entrenamiento como el error de generalización.
- La brecha entre ambos errores debe ser pequeña.

# Capacidad, subajuste y sobreajuste



# Capacidad, subajuste y sobreajuste



## Capacidad, subajuste y sobreajuste: No free lunch theorem



# Capacidad, subajuste y sobreajuste: No free lunch theorem

## "No Free Lunch" :(

D. H. Wolpert. The supervised learning no-free-lunch theorems. In Soft Computing and Industry, pages 25–42. Springer, 2002.

Our model is a simplification of reality



Simplification is based on assumptions (model bias)



Assumptions fail in certain situations

Roughly speaking:

***"No one model works best for all possible situations."***

Fuente: <https://analyticsindiamag.com/what-are-the-no-free-lunch-theorems-in-data-science/>

# Capacidad, subajuste y sobreajuste: Regularización

Por ejemplo, decaimiento del peso:

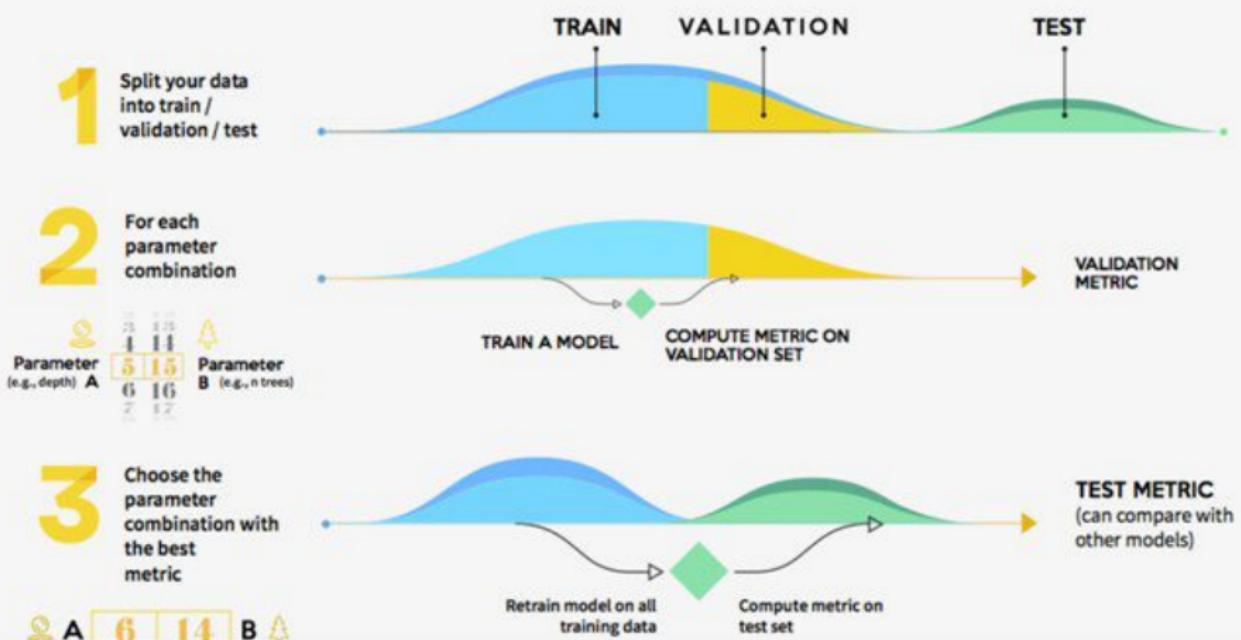
$$J(w) = MSE + \lambda w^T w$$

## Def.

Regularización es cualquier mecanismo que ayuda al agoritmo de aprendizaje a recudir su error de generalización.

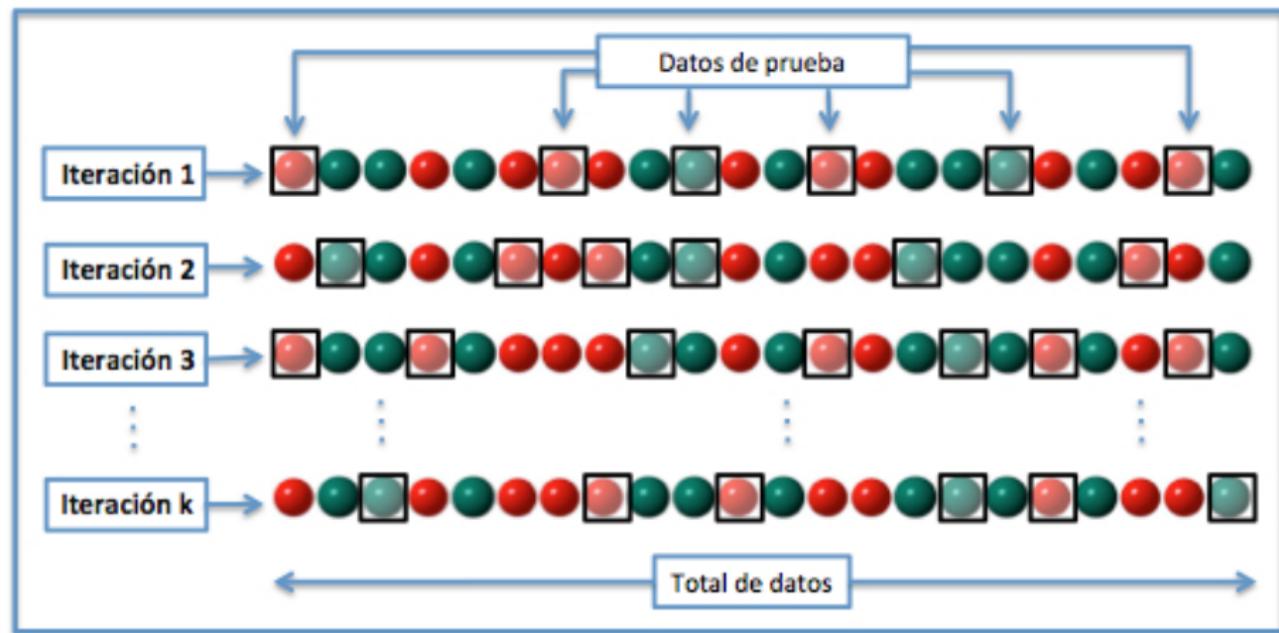
# Hiperparámetros y conjuntos de validación.

## HOLDOUT STRATEGY



Fuente: [https://www.kdnuggets.com/2017/08/  
dataiku-predictive-model-holdout-cross-validation.html](https://www.kdnuggets.com/2017/08/dataiku-predictive-model-holdout-cross-validation.html)

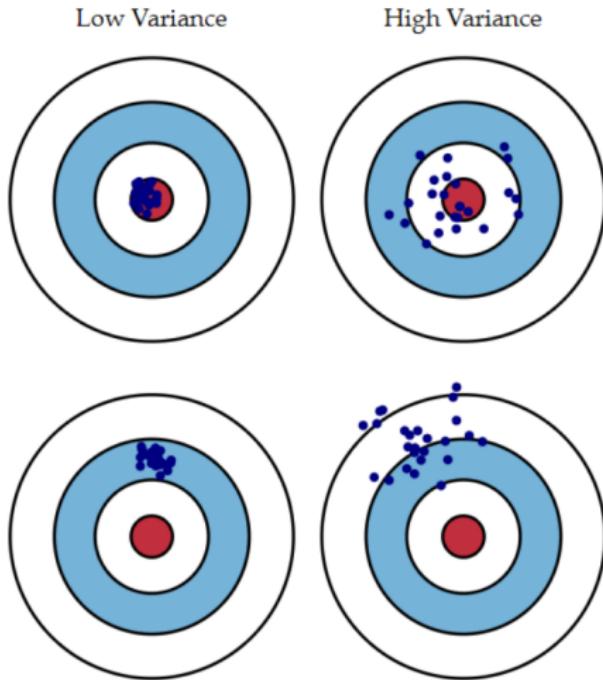
# Hiperparámetros y conjuntos de validación: validación cruzada



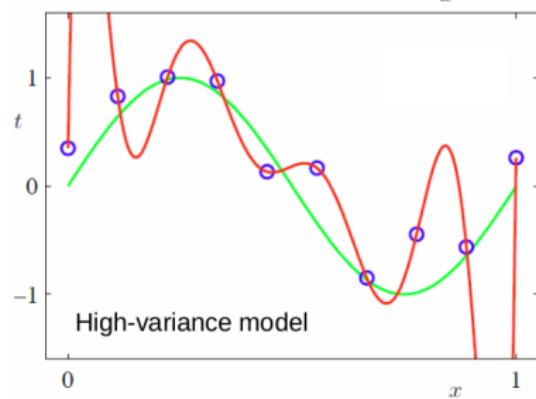
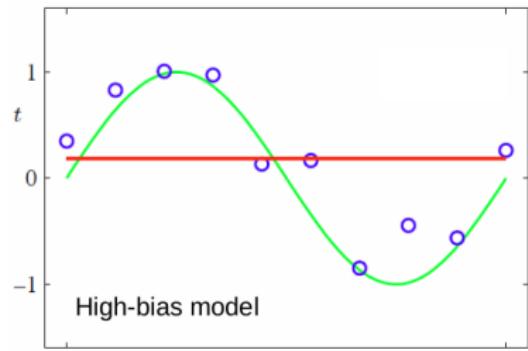
Fuente: [https://es.wikipedia.org/wiki/Validaci%C3%B3n\\_cruzada](https://es.wikipedia.org/wiki/Validaci%C3%B3n_cruzada)

# Estimadores, sesgo y varianza.

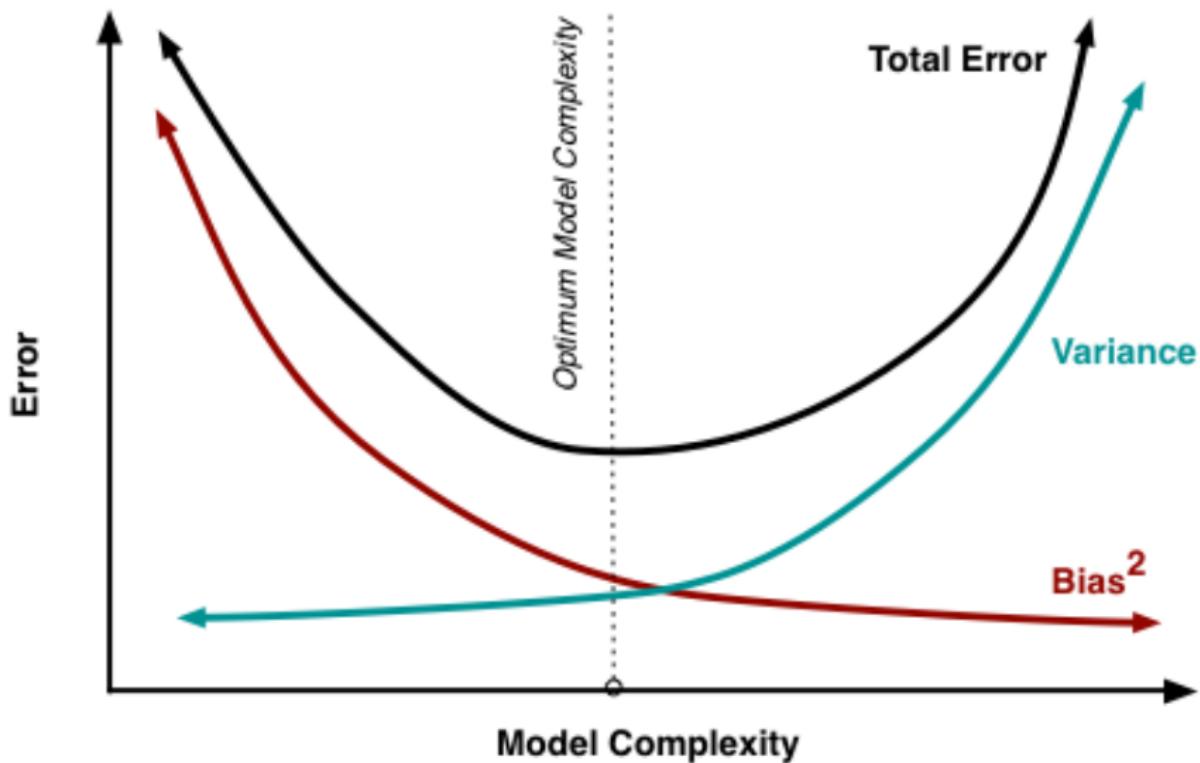
Low Bias



Scott Fortmann-Roe, Understanding the Bias-Variance Tradeoff, 2012



# Estimadores, sesgo y varianza.



# Estimadores, sesgo y varianza.

Visitar los siguientes enlaces:

- <http://scott.fortmann-roe.com/docs/BiasVariance.html>
- <https://ml.berkeley.edu/blog/posts/crash-course/part-4/>
- <https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/>

# Estimación de Máxima Verosimilitud(MLE).

- Estimación del Máximo Likelihood (MLE):

$$\ln \mathcal{L}(D, \theta) = \sum_{i=1}^n \ln f(x_i; \theta),$$

$$\theta_{MLE} = \arg \max(\mathcal{L}(\theta, D))$$

# Teorema de Bayes

Considerando funciones de densidad de probabilidad:

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}, \quad (1)$$

donde:

$$P(D) = \int_{\mathbb{R}^N} P(D|\theta)P(\theta)d\theta, \quad (2)$$

# Estadística Bayesiana.

- Estimación del A Posteriori (MAP) ó estimación de parámetros ó inferencia Bayesiana:

$$\theta_{MAP} = \arg \max(\mathcal{L}(\theta, D)P(\theta))$$

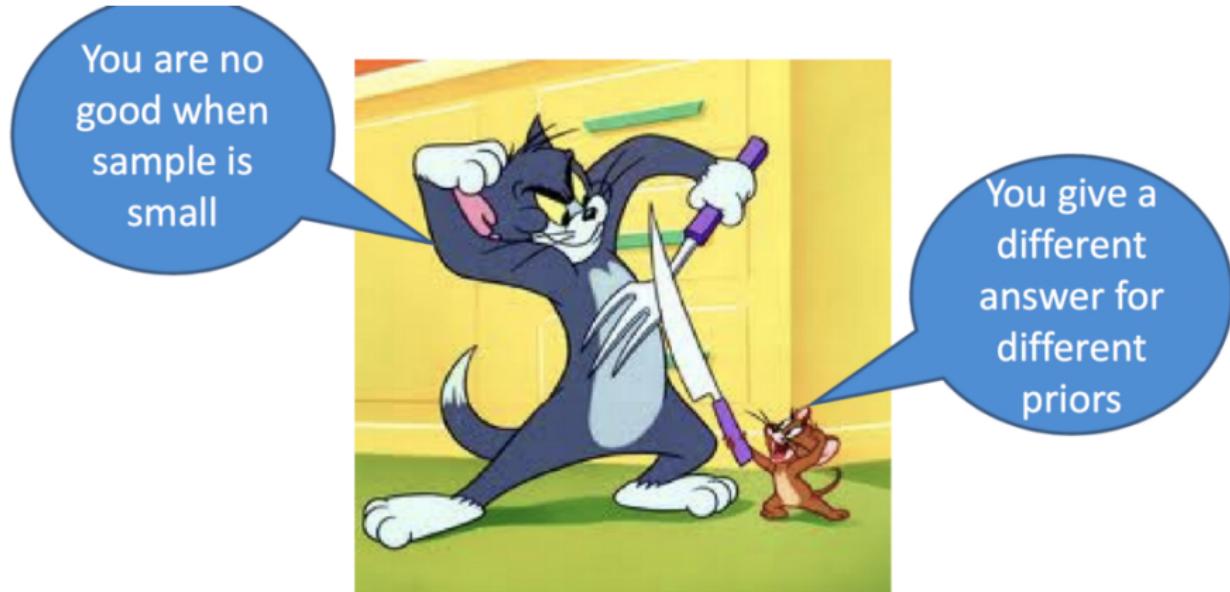
# Estadística Bayesiana.

- Estimación del A Posteriori (MAP) ó estimación de parámetros ó inferencia Bayesiana:

$$\theta_{MAP} = \arg \max(\mathcal{L}(\theta, D)P(\theta))$$

- Comparación de modelos (puede ser parte de la inferencia Bayesiana).

## Estadística frecuentista vs Bayesiana.



Fuente:<https://laptrinhx.com/maximum-likelihood-estimation-vs-maximum-a-posteriori-2539680111/>

## Estadística frecuentista vs Bayesiana.

Suppose we have data  $\mathcal{D} = \{x^{(i)}\}_{i=1}^N$

$$\boldsymbol{\theta}^{\text{MLE}} = \operatorname{argmax}_{\boldsymbol{\theta}} \prod_{i=1}^N p(\mathbf{x}^{(i)} | \boldsymbol{\theta})$$

Maximum Likelihood Estimate (MLE)

$$\boldsymbol{\theta}^{\text{MAP}} = \operatorname{argmax}_{\boldsymbol{\theta}} \prod_{i=1}^N p(\mathbf{x}^{(i)} | \boldsymbol{\theta}) p(\boldsymbol{\theta})$$

Maximum a posteriori (MAP) estimate

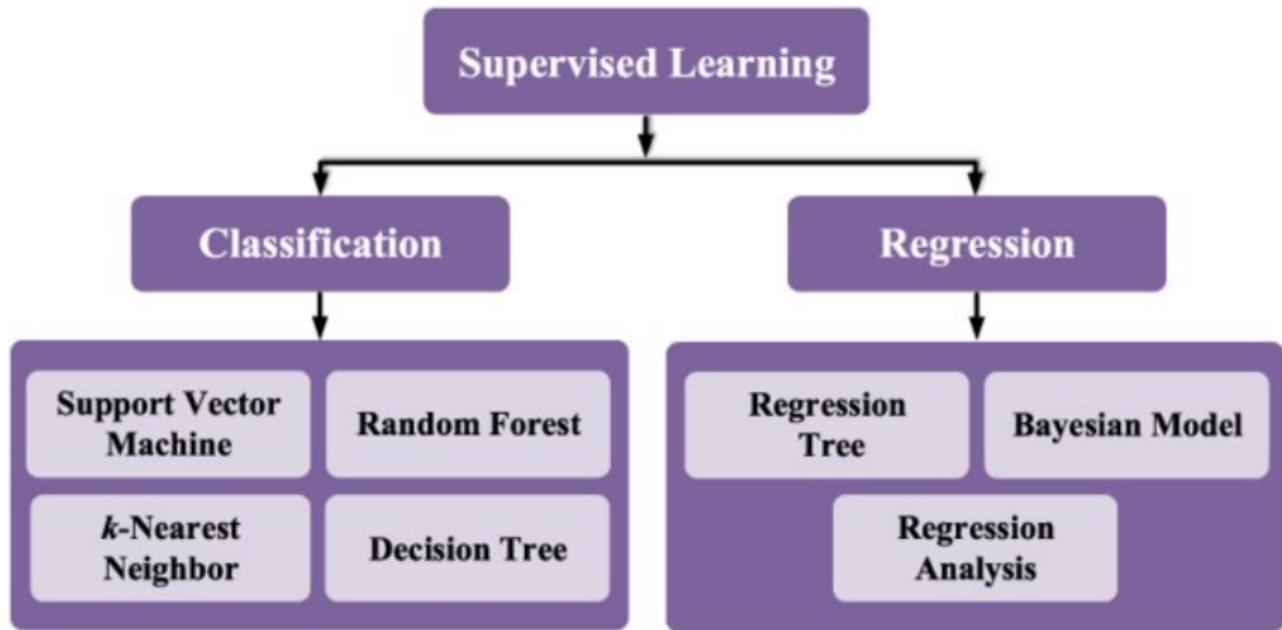


Prior

Fuente: <https://medium.com/@tzjy/>

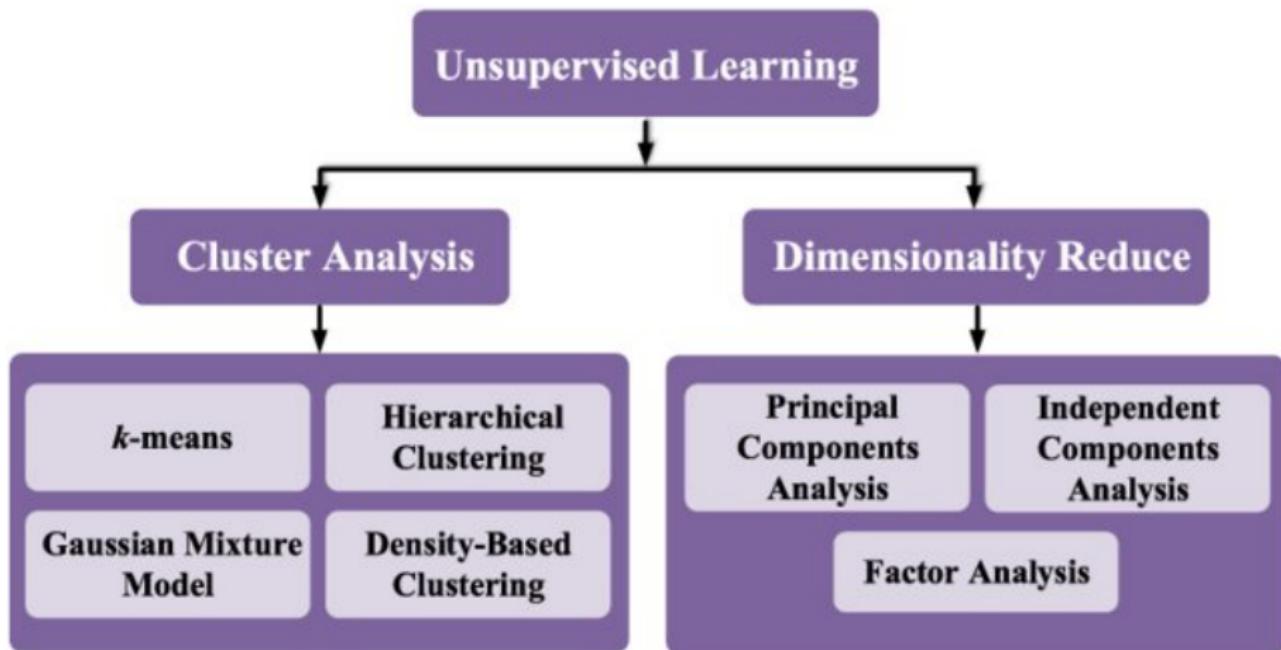
whats-the-difference-between-maximum-likelihood-estimation-mle-and-maximum

# Algoritmos de aprendizaje supervisado.



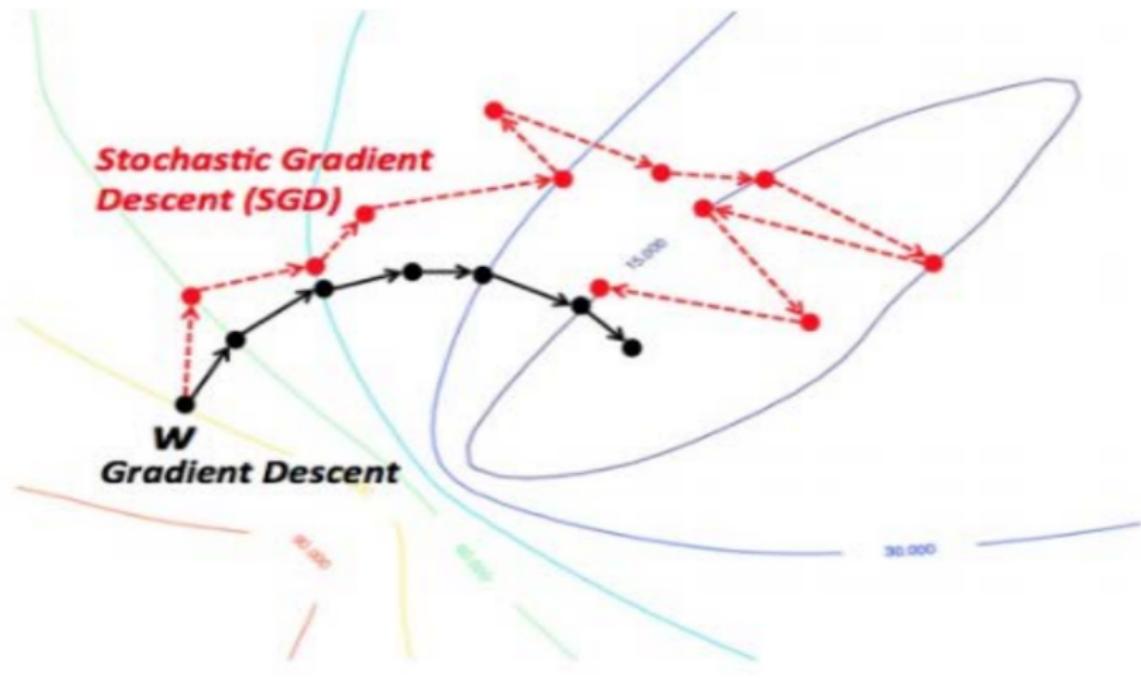
Fuente: arXiv:2003.10146

# Algoritmos de aprendizaje no supervisado.



Fuente: arXiv:2003.10146

# Descenso del gradiente estocástico.



Fuente: <https://www.slideshare.net/microlife/from-neural-networks-to-deep-learning>

Dra. Lili Guadarrama Bustos Dr. Isidro Gómez

# Descenso del gradiente estocástico.

## Tarea propuesta

Programar un descenso del gradiente estocástico y minimizar una función con él.

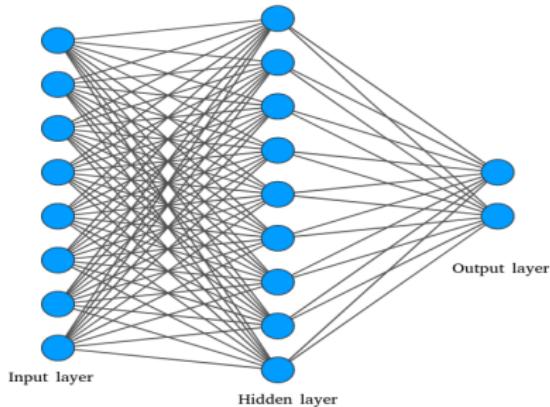
# Retos que motivan al Aprendizaje Profundo.

- Curso de la dimensionalidad.
- Local Constancy and Smoothness Regularization.
- Manifold Learning.

## Clase 2: Introducción al Aprendizaje Profundo

- Inicia la parte II del libro de referencia.
- Nos centraremos en el capítulo 6.

# Perceptrón multicapa



Deep feedforward networks /feedforward neural networks

$$f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$$

$f^{(i)}$  es llamada la i-ésima pared de la red.  $f(x)$  debe coincidir con  $f^*(x)$ , cada ejemplo  $x$  tiene asociada una etiqueta  $y = f^*(x)$

# Perceptrón multicapa

Para que obtener un algoritmo no-lineal, hay que emplear una función  $\phi(x)$  para modificar las entradas a las funciones.

## Truco del kernel (descrito en 5.7.2)

$$w^T x + b = b \sum_{i=1}^m \alpha_i x^T x^{(i)}$$

se puede cambiar  $x - > \phi(x)$  y el producto punto por un kernel  $k(x, x^{(i)}) = \phi(x) \dot{\phi}(x^{(i)})$ . Entonces se pueden hacer predicciones mediante:

$$f(x) = b + \sum_{i=1}^m \alpha_i k(x, x^{(i)})$$

donde la función es no-lineal respecto a  $x$ , pero la relación entre  $\phi(x)$  y  $f(x)$  es lineal. También entre  $\alpha$  y  $f(x)$  es lineal.

## Truco del kernel (descrito en 5.7.2)

La función basada en kernel es exactamente equivalente a preprocesar los datos aplicando  $\phi(x)$  a todas las entradas, y luego aprender un modelo lineal en el nuevo espacio transformado.

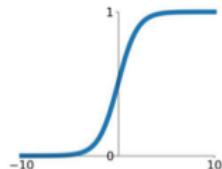
Ventajas:

- ① Permite aprender modelos que son funciones no lineales de  $x$  usando técnicas de optimización convexas.
- ② La función kernel, generalmente, permite una implementación que es significativamente más eficiente, en términos computacionales, que construir dos vectores  $\phi(x)$  y luego tomar explícitamente sus productos.

# Funciones de activación

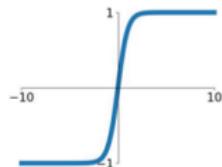
## Sigmoid

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



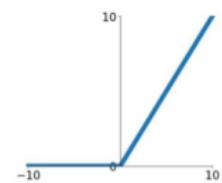
## tanh

$$\tanh(x)$$



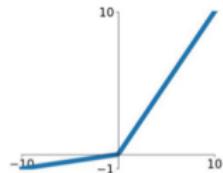
## ReLU

$$\max(0, x)$$



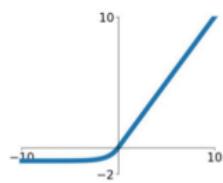
## Leaky ReLU

$$\max(0.1x, x)$$



## ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



## Multilayer Feedforward Networks are Universal Approximators

KURT HORNICK

Technische Universität Wien

MAXWELL STINCHCOMBE AND HALBERT WHITE

University of California, San Diego

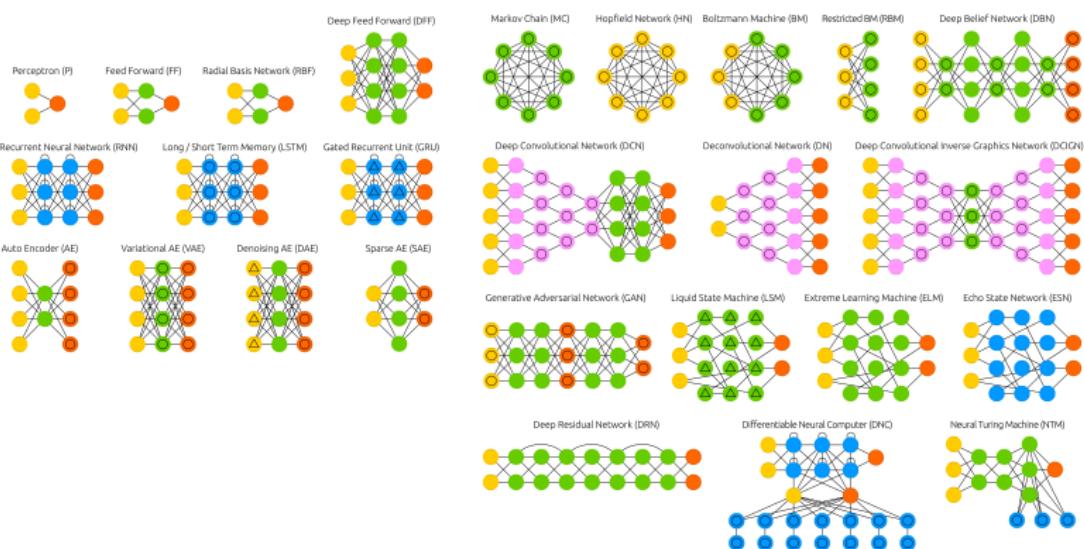
(Received 16 September 1988; revised and accepted 9 March 1989)

**Abstract**—This paper rigorously establishes that standard multilayer feedforward networks with as few as one hidden layer using arbitrary squashing functions are capable of approximating any Borel measurable function from one finite dimensional space to another to any desired degree of accuracy, provided sufficiently many hidden units are available. In this sense, multilayer feedforward networks are a class of universal approximators.

**Keywords**—Feedforward networks, Universal approximation, Mapping networks, Network representation capability, Stone-Weierstrass Theorem, Squashing functions, Sigma-Pi networks, Back-propagation networks.

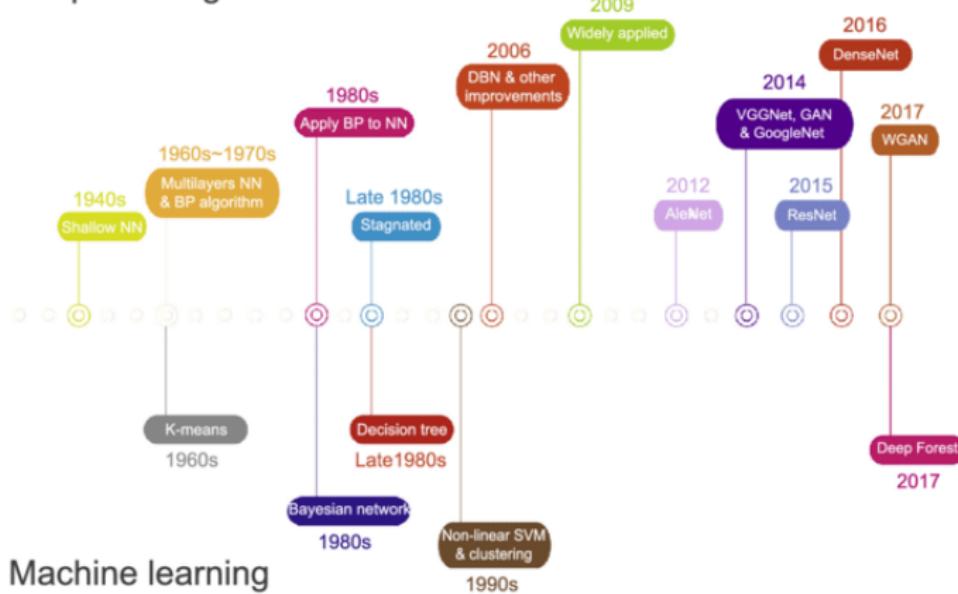
# Múltiples arquitecturas

- Input Cell
- Backfied Input Cell
- △ Noisy Input Cell
- Hidden Cell
- Probabilistic Hidden Cell
- △ Spiking Hidden Cell
- Capsule Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- △ Gated Memory Cell
- Kernel
- Convolution or Pool



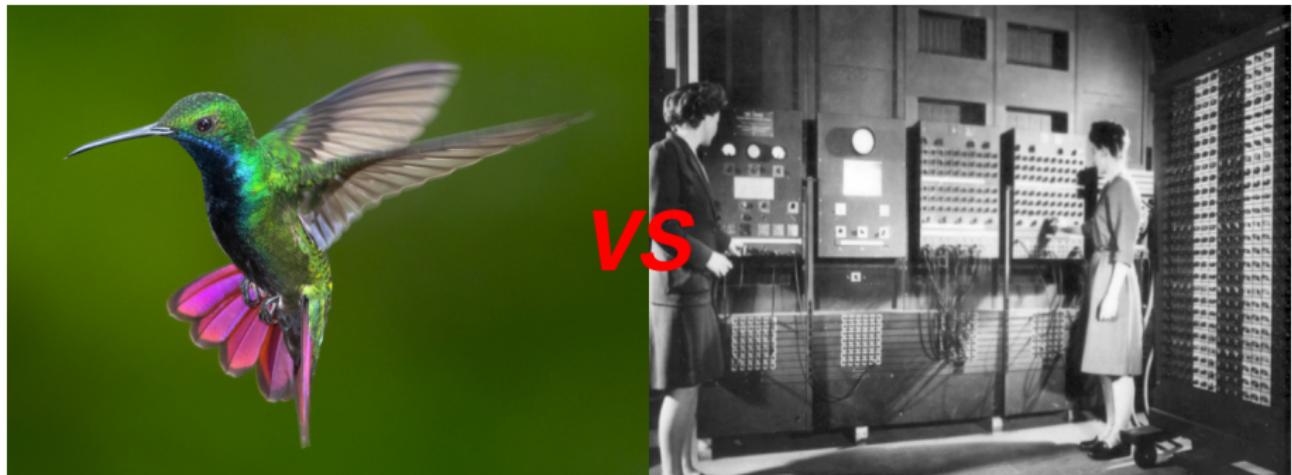
# Cronología ML

## Deep learning

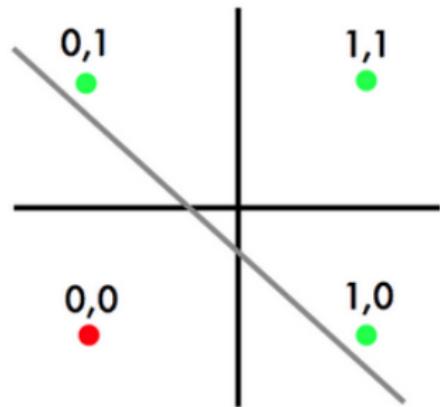


# Aprendizaje máquina

1940s



# El problema de la compuerta XOR



OR



XOR

Ver notebooks: <https://github.com/igomezv/DLCIMATAGS/tree/main/notebooks/clase%202>

# Gradient-based learning

Como preámbulo: repasar secciones 4.3 y 5.9, correspondientes al aprendizaje por medio del descenso del gradiente y al descenso del gradiente estocástico, respectivamente.

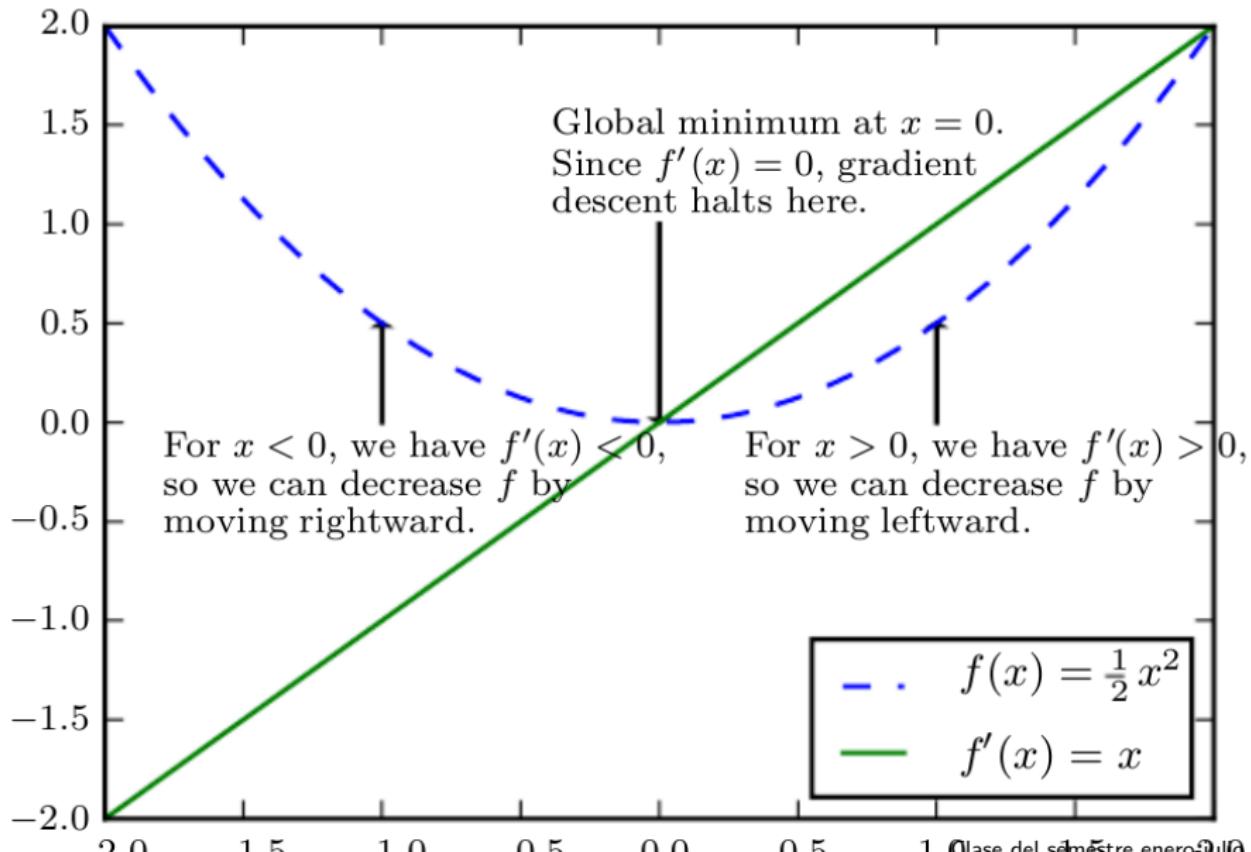
# Aprendizaje basado en descenso del gradiente

## Optimización:

Maximizar o minimizar una función  $f(x)$ .

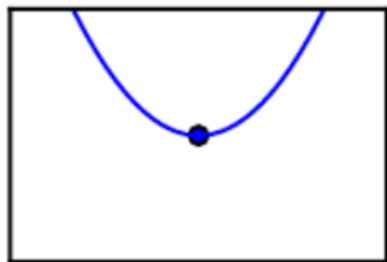
- La función a maximizar o minimizar se conoce como *función objetivo* o *criterio*.
- Cuando esta función se está minimizando, se le suele nombrar *función de costo*, *función de pérdida* o *función de error*.

# Aprendizaje basado en descenso del gradiente

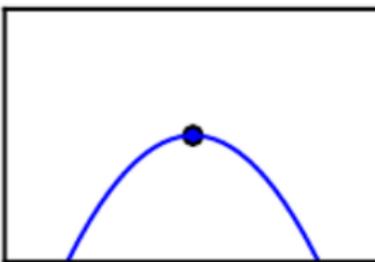


# Aprendizaje basado en descenso del gradiente

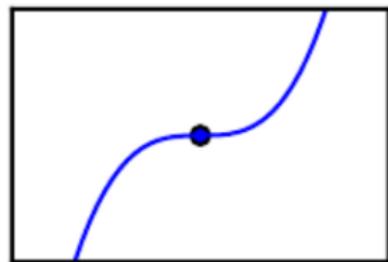
Minimum



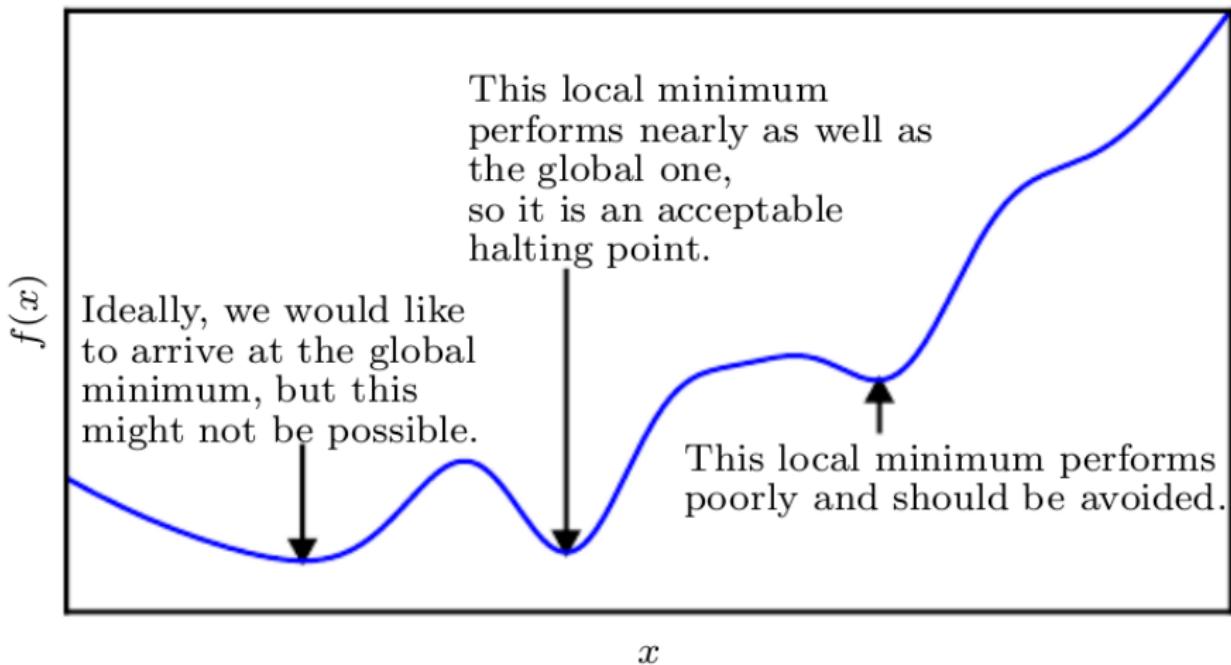
Maximum



Saddle point



# Aprendizaje basado en descenso del gradiente



# Descenso del gradiente

$$x' = x - \epsilon \nabla_x f(x)$$

# Descenso del gradiente estocástico

En el descenso del gradiente, si:

$$\nabla_{\theta} J(\theta) = \frac{1}{m} \sum_{i=1}^m \nabla_{\theta} L(x^{(i)}, y^{(i)}, \theta)$$

entonces el costo computacional del cálculo es  $O(m)$ .

# Descenso del gradiente estocástico

Sea  $B = x^{(1)}, \dots, x^{(m')}$  con  $x^{(i)}$  extraídos uniformemente del conjunto de entrenamiento.  $B$  se conoce como *mini-batch* con tamaño  $m'$ , regularmente  $1 < m' < \sim 1000$ . Entonces:

$$g = \frac{1}{m'} \sum_{i=1}^{m'} \nabla_{\theta} L(x^{(i)}, y^{(i)}, \theta)$$

y

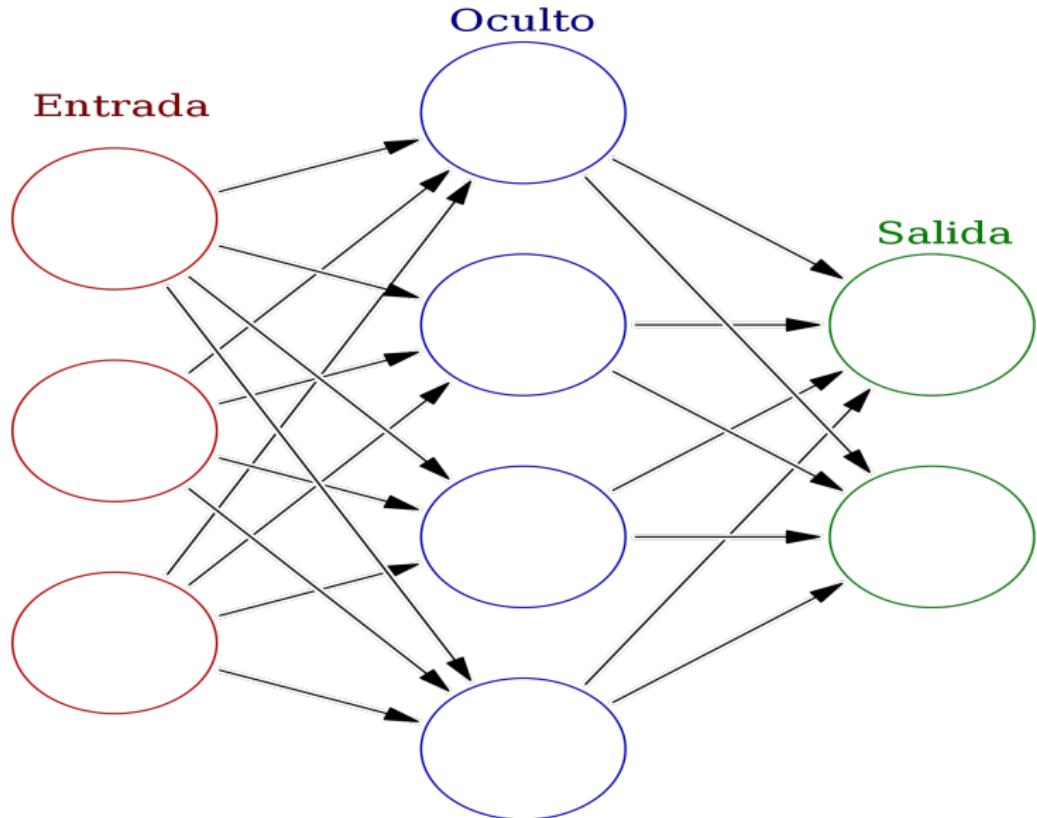
$$\theta \leftarrow \theta - \epsilon g$$

## Descenso del gradiente estocástico

En general, el descenso del gradiente se ha considerado lento o poco fiable. En el pasado, la aplicación del descenso de gradiente a problemas de optimización no convexos se consideraba temeraria o sin principios. Hoy en día, sabemos que los modelos de aprendizaje automático funcionan muy bien cuando se entranan con el descenso de gradiente.

El algoritmo de optimización puede no estar garantizado para llegar incluso a un mínimo local en un tiempo razonable, pero a menudo encuentra un valor muy bajo de la función de coste lo suficientemente rápido como para ser útil.

# Unidades ocultas



# Unidades ocultas

**¿Cómo elegir el tipo de unidades ocultas en las capas intermedias?**

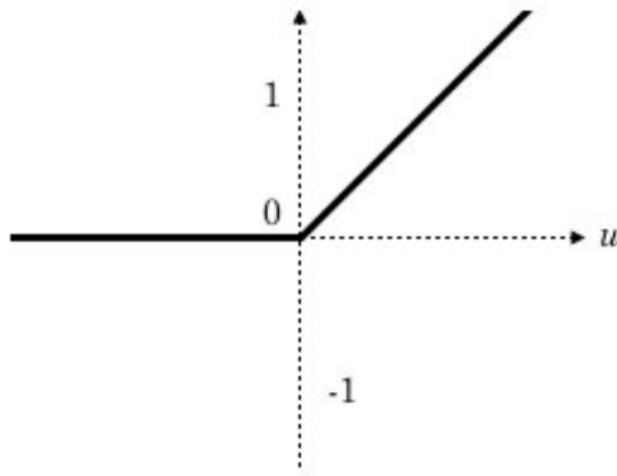
Se trata de un área de investigación muy activa y que todavía no está guiada por principios teóricos.

Recordemos...

$$h = g(W^T x + b)$$

# Rectified Linear Unit (ReLU)

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



# Generalizaciones de ReLU

$$h_i = g(z, \alpha)_i = \max(0, z_i) + \alpha_i \min(0, z_i)$$

- Absolute value rectification:  $\alpha_i = -1 \implies g(z) = |z|$ .

# Generalizaciones de ReLU

$$h_i = g(z, \alpha)_i = \max(0, z_i) + \alpha_i \min(0, z_i)$$

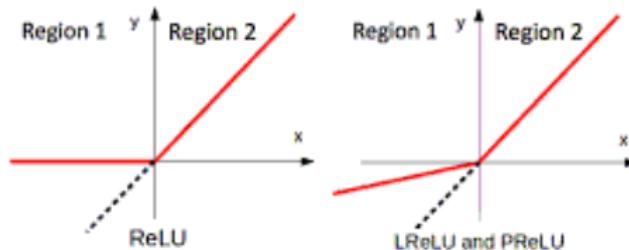
- Absolute value rectification:  $\alpha_i = -1 \implies g(z) = |z|$ .
- Leaky ReLU:  $\alpha_i$  tiene valores cercanos a 0,01.

# Generalizaciones de ReLU

$$h_i = g(z, \alpha)_i = \max(0, z_i) + \alpha_i \min(0, z_i)$$

- Absolute value rectification:  $\alpha_i = -1 \implies g(z) = |z|$ .
- Leaky ReLU:  $\alpha_i$  tiene valores cercanos a 0,01.
- Parametric ReLU:  $\alpha$  es un parámetro a aprender.

# Generalizaciones de ReLU



## Unidades Maxout

$$g(z)_i = \max z_j, j \in G^{(i)}$$

where  $G^{(i)}$  is the set of indices into the inputs for group  $i$ ,  
 $(i-1)k+1, \dots, i_k$ . A mayor valor  $k$ , unidades maxout pueden aprender a  
aproximar cualquier función convexa con fidelidad arbitraria.

# Unidades Maxout

$$g(z)_i = \max z_j, j \in G^{(i)}$$

where  $G^{(i)}$  is the set of indices into the inputs for group  $i$ ,  
 $(i - 1)k + 1, \dots, i_k$ . A mayor valor  $k$ , unidades maxout pueden aprender a  
aproximar cualquier función convexa con fidelidad arbitraria.

Ayudan a...

Evitar el olvido catastrófico.

# Unidades Maxout

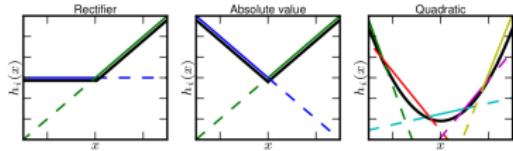


Figure 1. Graphical depiction of how the maxout activation function can implement the rectified linear, absolute value rectifier, and approximate the quadratic activation function. This diagram is 2D and only shows how maxout behaves with a 1D input, but in multiple dimensions a maxout unit can approximate arbitrary convex functions.

# Sigmoide logístico y tangente hiperbólica

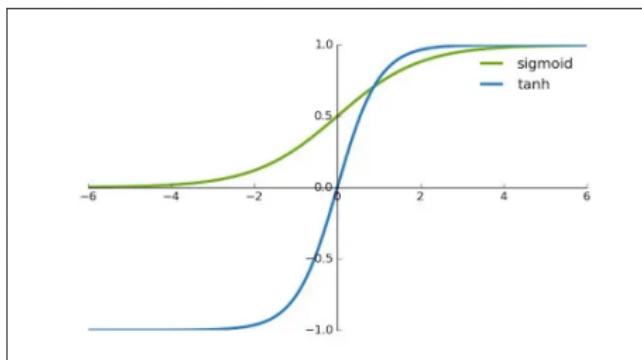
Antes de existir ReLu, la mayoría de las redes neuronales usaban la función sigmoide logístico como función de activación

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

o la función tangente hiperbólica

$$g(z) = \tanh(z)$$

# Sigmoide logístico y tangente hiperbólica



## Otras unidades ocultas: ventajas de función lineal

Consideremos una red neuronal con algunas paredes con  $n$  entradas y  $p$  salidas.  $h = g(W^T x + b)$ . Sean  $U, V$  son dos matrices de pesos, si la primera no tiene función de activación se puede sustituir  $h = g(V^T U^T x + b)$ .

## Otras unidades ocultas: ventajas de función lineal

Consideremos una red neuronal con algunas paredes con  $n$  entradas y  $p$  salidas.  $h = g(W^T x + b)$ . Sean  $U, V$  son dos matrices de pesos, si la primera no tiene función de activación se puede sustituir  $h = g(V^T U^T x + b)$ . Si  $U$  produce  $q$  salidas, entonces juntas contienen  $(n + p)q$  parámetros, mientras que  $W$  tendría  $np$ .

## Otro tipo de unidades ocultas

- Función de base radial:  $h_i = \exp\left(-\frac{1}{\sigma_i^2} \|W_i(i) - x\|^2\right)$ . Más activa cuando  $x$  se acerca a un modelo  $W_i(i)$ . Satura en 0 para la mayoría de los casos, es difícil de optimizar.
- Softplus:  $g(a) = \log(1 + e^a)$  (versión suavizada de un rectificador). Recomendada en la última capa, es mejor ReLU en capas ocultas.
- Hard tanh:  $g(a) = \max((-1, \min(1, a)))$ , similar a tanh y ReLU, pero a diferencia de ReLU, está acotada.

## Otro tipo de unidades ocultas

- Función de base radial:  $h_i = \exp\left(-\frac{1}{\sigma_i^2} \|W_i(i) - x\|^2\right)$ . Más activa cuando  $x$  se acerca a un modelo  $W_i(i)$ . Satura en 0 para la mayoría de los casos, es difícil de optimizar.
- Softplus:  $g(a) = \log(1 + e^a)$  (versión suavizada de un rectificador). Recomendada en la última capa, es mejor ReLU en capas ocultas.
- Hard tanh:  $g(a) = \max((-1, \min(1, a)))$ , similar a tanh y ReLU, pero a diferencia de ReLU, está acotada.

### ¿Nuevos tipos de unidades ocultas?

Hay gran variedad comparables con los tipos conocidos y son tan comunes que no resultan interesantes.

# Diseño de arquitectura

Primera capa:

$$h^{(1)} = g^{(1)}(W^{(1)T}x + b^{(1)}),$$

# Diseño de arquitectura

Primera capa:

$$h^{(1)} = g^{(1)}(W^{(1)T}x + b^{(1)}),$$

la segunda capa:

$$h^{(2)} = g^{(2)}(W^{(2)T}x + b^{(2)}),$$

# Diseño de arquitectura

Primera capa:

$$h^{(1)} = g^{(1)}(W^{(1)T}x + b^{(1)}),$$

la segunda capa:

$$h^{(2)} = g^{(2)}(W^{(2)T}x + b^{(2)}),$$

y así, sucesivamente.

# Diseño de arquitectura

Las redes más profundas suelen ser capaces de utilizar muchas menos unidades por capa y muchos menos parámetros y suelen generalizar en el conjunto de pruebas, pero también suelen ser más difíciles de optimizar. La arquitectura de red ideal para una tarea debe encontrarse mediante la experimentación guiada por el control del error del conjunto de validación.

# Diseño de arquitectura

El teorema de aproximación universal establece que una red feedforward con una capa de salida lineal capa de salida lineal y al menos una capa oculta con cualquier función de activación (como la función de activación sigmoide logística) puede aproximar cualquier función de Borel medible de un espacio finito a otro con cualquier cantidad de error no nula deseada, siempre que la red tenga suficientes unidades ocultas.

# Diseño de arquitectura

El teorema de aproximación universal establece que una red feedforward con una capa de salida lineal capa de salida lineal y al menos una capa oculta con cualquier función de activación (como la función de activación sigmoide logística) puede aproximar cualquier función de Borel medible de un espacio finito a otro con cualquier cantidad de error no nula deseada, siempre que la red tenga suficientes unidades ocultas.

## Función de Borel

Cualquier función continua sobre un subconjunto cerrado y acotado de  $R^n$  es Borel medible y, por tanto, puede ser aproximada por una red neuronal.

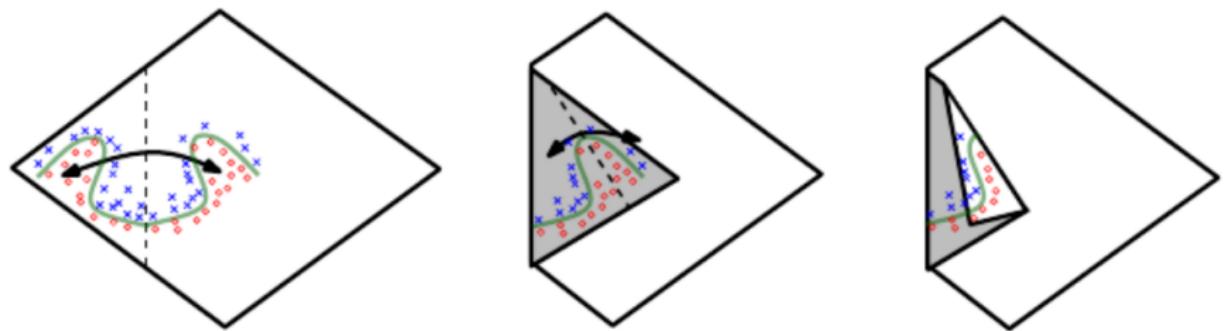
## Diseño de arquitectura

Las redes lineales a trozos (que pueden obtenerse de ReLU y sus variantes o las unidades maxout) pueden representar funciones con un número de regiones que es exponencial en la profundidad de la red.

## Diseño de arquitectura

Las redes lineales a trozos (que pueden obtenerse de ReLU y sus variantes o las unidades maxout) pueden representar funciones con un número de regiones que es exponencial en la profundidad de la red.

# Diseño de arquitectura: más profundo, mejor



# Diseño de arquitectura

La elección de un modelo profundo codifica un modelo muy general de que la función que queremos aprender debe implicar la composición de varias funciones más simples.

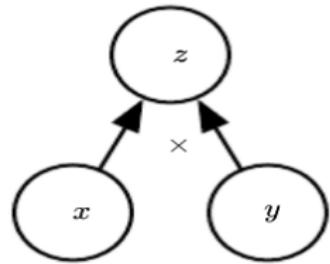
# Diseño de arquitectura

<https://playground.tensorflow.org>

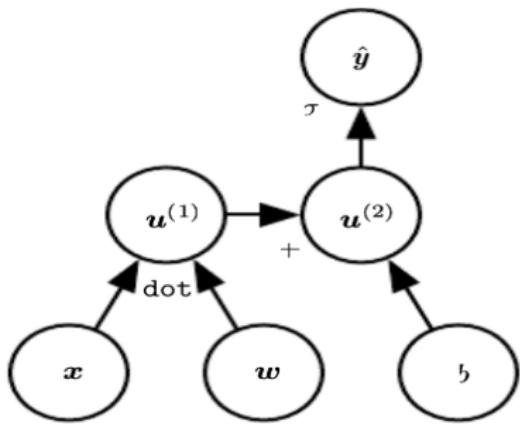
# Backpropagation y otros algoritmos de diferenciación

Backpropagation solo se refiere a calcular el gradiente, mientras que otro algoritmo, que podría ser del tipo del descenso del gradiente, se usa para llevar a cabo el aprendizaje de este gradiente.

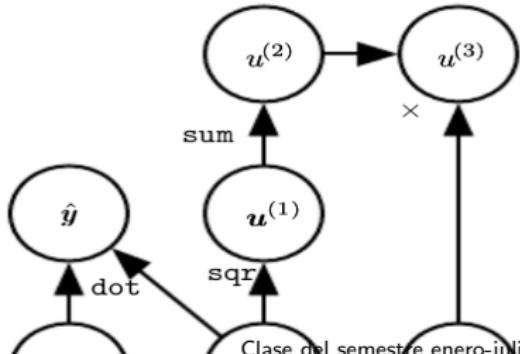
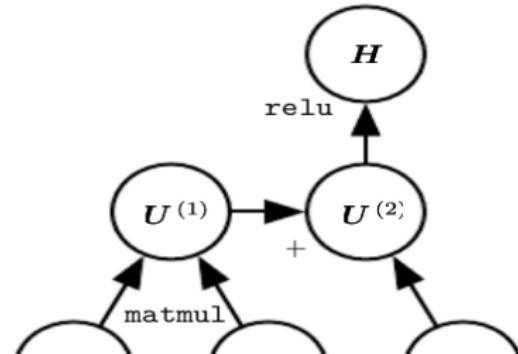
# Grafos computacionales



(a)



(b)



# Backpropagation y regla de la cadena

Sea  $y = g(x)$ ,  $z = f(g(x)) = f(y)$

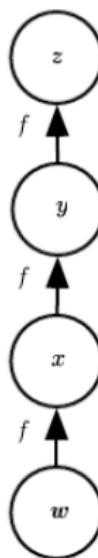
$$\frac{dz}{dx} = \frac{dz}{dy}$$

Sean  $x \in \mathbb{R}^m$ ,  $y \in \mathbb{R}^n$ ,  $g : \mathbb{R}^m \rightarrow \mathbb{R}^n$ ,  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ , si  $y = g(x)$  y  $z = f(y)$ :

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

En forma vectorial:  $\nabla_x z = (\frac{\partial y}{\partial x})^T \nabla_y z$ , donde  $(\frac{\partial y}{\partial x}) = J_y \in \mathbb{M}^{n \times m}$ .

# Subexpresiones repetidas



$$\begin{aligned}\frac{\partial z}{\partial w} &= \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \frac{\partial x}{\partial w} \\ &= f'(y) f'(x) f'(w) \\ &= f'(f(f(w))) f'(f(w)) f'(w)\end{aligned}$$

# Propagación hacia adelante general

---

**Algorithm 6.1** A procedure that performs the computations mapping  $n_i$  inputs  $u^{(1)}$  to  $u^{(n_i)}$  to an output  $u^{(n)}$ . This defines a computational graph where each node computes numerical value  $u^{(i)}$  by applying a function  $f^{(i)}$  to the set of arguments  $\mathbb{A}^{(i)}$  that comprises the values of previous nodes  $u^{(j)}$ ,  $j < i$ , with  $j \in Pa(u^{(i)})$ . The input to the computational graph is the vector  $\mathbf{x}$ , and is set into the first  $n_i$  nodes  $u^{(1)}$  to  $u^{(n_i)}$ . The output of the computational graph is read off the last (output) node  $u^{(n)}$ .

---

```
for  $i = 1, \dots, n_i$  do
     $u^{(i)} \leftarrow x_i$ 
end for
for  $i = n_i + 1, \dots, n$  do
     $\mathbb{A}^{(i)} \leftarrow \{u^{(j)} \mid j \in Pa(u^{(i)})\}$ 
     $u^{(i)} \leftarrow f^{(i)}(\mathbb{A}^{(i)})$ 
end for
return  $u^{(n)}$ 
```

---

# Backpropagation simple

---

**Algorithm 6.2** Simplified version of the back-propagation algorithm for computing the derivatives of  $u^{(n)}$  with respect to the variables in the graph. This example is intended to further understanding by showing a simplified case where all variables are scalars, and we wish to compute the derivatives with respect to  $u^{(1)}, \dots, u^{(n)}$ .

---

Run forward propagation (algorithm 6.1 for this example) to obtain the activations of the network

Initialize `grad_table`, a data structure that will store the derivatives that have been computed. The entry `grad_table[u(i)]` will store the computed value of  $\frac{\partial u^{(n)}}{\partial u^{(i)}}$ .

```
grad_table[u(n)] ← 1
for j = n - 1 down to 1 do
    The next line computes  $\frac{\partial u^{(n)}}{\partial u^{(j)}} = \sum_{i:j \in Pa(u^{(i)})} \frac{\partial u^{(n)}}{\partial u^{(i)}} \frac{\partial u^{(i)}}{\partial u^{(j)}}$  using stored values:
    grad_table[u(j)] ←  $\sum_{i:j \in Pa(u^{(i)})} \text{grad\_table}[u^{(i)}] \frac{\partial u^{(i)}}{\partial u^{(j)}}$ 
end for
return {grad_table[u(i)] | i = 1, ..., ni}
```

---

# Propagación hacia adelante en un Perceptrón Multicapa

---

**Algorithm 6.3** Forward propagation through a typical deep neural network and the computation of the cost function. The loss  $L(\hat{\mathbf{y}}, \mathbf{y})$  depends on the output  $\hat{\mathbf{y}}$  and on the target  $\mathbf{y}$  (see section 6.2.1.1 for examples of loss functions). To obtain the total cost  $J$ , the loss may be added to a regularizer  $\Omega(\theta)$ , where  $\theta$  contains all the parameters (weights and biases). Algorithm 6.4 shows how to compute gradients of  $J$  with respect to parameters  $\mathbf{W}$  and  $\mathbf{b}$ . For simplicity, this demonstration uses only a single input example  $\mathbf{x}$ . Practical applications should use a minibatch. See section 6.5.7 for a more realistic demonstration.

---

**Require:** Network depth,  $l$

**Require:**  $\mathbf{W}^{(i)}, i \in \{1, \dots, l\}$ , the weight matrices of the model

**Require:**  $\mathbf{b}^{(i)}, i \in \{1, \dots, l\}$ , the bias parameters of the model

**Require:**  $\mathbf{x}$ , the input to process

**Require:**  $\mathbf{y}$ , the target output

$$\mathbf{h}^{(0)} = \mathbf{x}$$

**for**  $k = 1, \dots, l$  **do**

$$\mathbf{a}^{(k)} = \mathbf{b}^{(k)} + \mathbf{W}^{(k)} \mathbf{h}^{(k-1)}$$

$$\mathbf{h}^{(k)} = f(\mathbf{a}^{(k)})$$

**end for**

$$\hat{\mathbf{y}} = \mathbf{h}^{(l)}$$

$$J = L(\hat{\mathbf{y}}, \mathbf{y}) + \lambda \Omega(\theta)$$

# Backpropagation en un Perceptrón Multicapa

---

**Algorithm 6.4 Backward** computation for the deep neural network of algorithm 6.3, which uses in addition to the input  $\mathbf{x}$  a target  $\mathbf{y}$ . This computation yields the gradients on the activations  $\mathbf{a}^{(k)}$  for each layer  $k$ , starting from the output layer and going backwards to the first hidden layer. From these gradients, which can be interpreted as an indication of how each layer's output should change to reduce error, one can obtain the gradient on the parameters of each layer. The gradients on weights and biases can be immediately used as part of a stochastic gradient update (performing the update right after the gradients have been computed) or used with other gradient-based optimization methods.

---

After the forward computation, compute the gradient on the output layer:

$\mathbf{g} \leftarrow \nabla_{\hat{\mathbf{y}}} J = \nabla_{\hat{\mathbf{y}}} L(\hat{\mathbf{y}}, \mathbf{y})$   
**for**  $k = l, l - 1, \dots, 1$  **do**

Convert the gradient on the layer's output into the gradient into the pre-nonlinearity activation (element-wise multiplication if  $f$  is element-wise):

$$\mathbf{g} \leftarrow \nabla_{\mathbf{a}^{(k)}} J = \mathbf{g} \odot f'(\mathbf{a}^{(k)})$$

Compute gradients on weights and biases (including the regularization term, where needed):

$$\nabla_{\mathbf{b}^{(k)}} J = \mathbf{g} + \lambda \nabla_{\mathbf{b}^{(k)}} \Omega(\theta)$$

$$\nabla_{\mathbf{W}^{(k)}} J = \mathbf{g} \mathbf{h}^{(k-1)\top} + \lambda \nabla_{\mathbf{W}^{(k)}} \Omega(\theta)$$

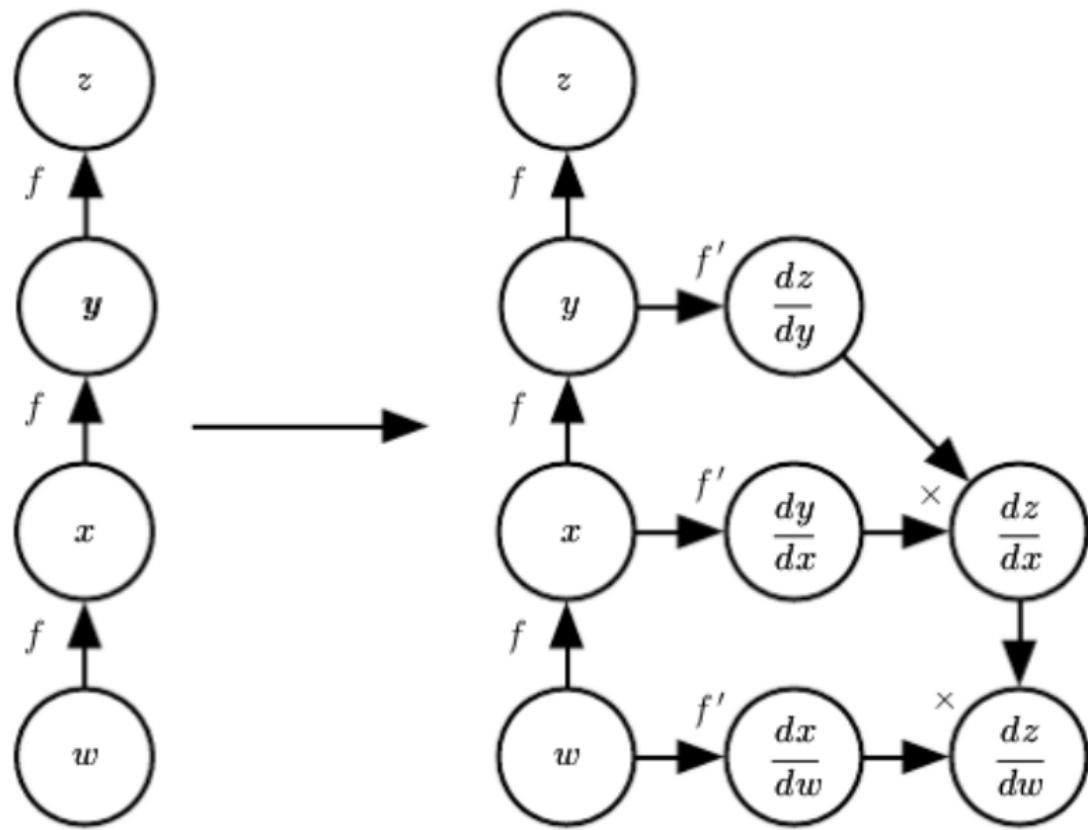
Propagate the gradients w.r.t. the next lower-level hidden layer's activations:

$$\mathbf{g} \leftarrow \nabla_{\mathbf{h}^{(k-1)}} J = \mathbf{W}^{(k)\top} \mathbf{g}$$

**end for**

---

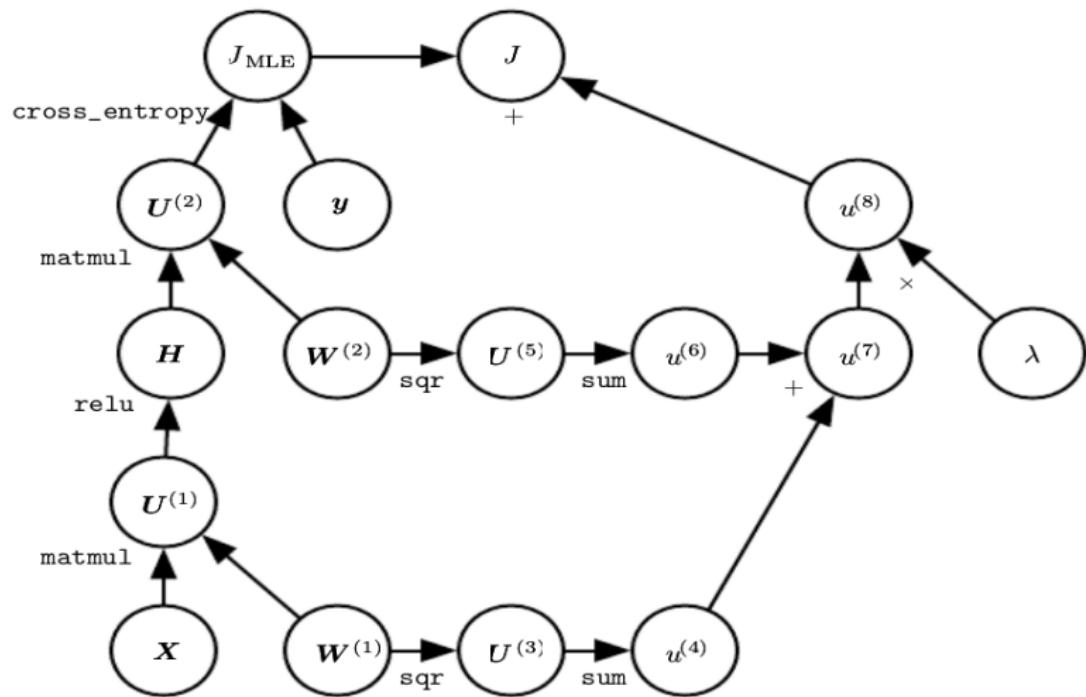
## Símbolo a símbolo / símbolo a número



## Ejemplo: Representación de un MLP

1 capa oculta, relu, weight decay, cross entropy:

$$J = J_{\text{MLE}} + \lambda \left( \sum_{i,j} (W_{i,j}^{(1)})^2 + \sum_{i,j} (W_{i,j}^{(2)})^2 \right)$$



# Regularización para AP

La regularización es cualquier modificación que se realiza en el algoritmo de aprendizaje para reducir el error de generalización.

mayor sesgo  $\leftrightarrow$  menor varianza

Idea:

El mejor modelo ajuste es aquel modelo general regularizado  
adecuadamente

## Parameter norm penalties

Sea  $J = J(\theta; X, y)$  la función objetivo, denotamos como  $\tilde{J}$  a la función objetivo regularizada definida como

$$\tilde{J}((\theta; X, y) = J(\theta; X, y) + \alpha \Gamma(\theta)$$

- $\alpha \geq 0$
- $\Gamma$  norm penalty

En AP se utilizaran normas que penalicen solamente los pesos en cada capa dejando los sesgos sin regularizar (evitando que aumente la varianza & underfitting)

weight decay = norma  $L^2$

# Inferencia bayesiana

<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8699938/>