



Aprendizaje Automático

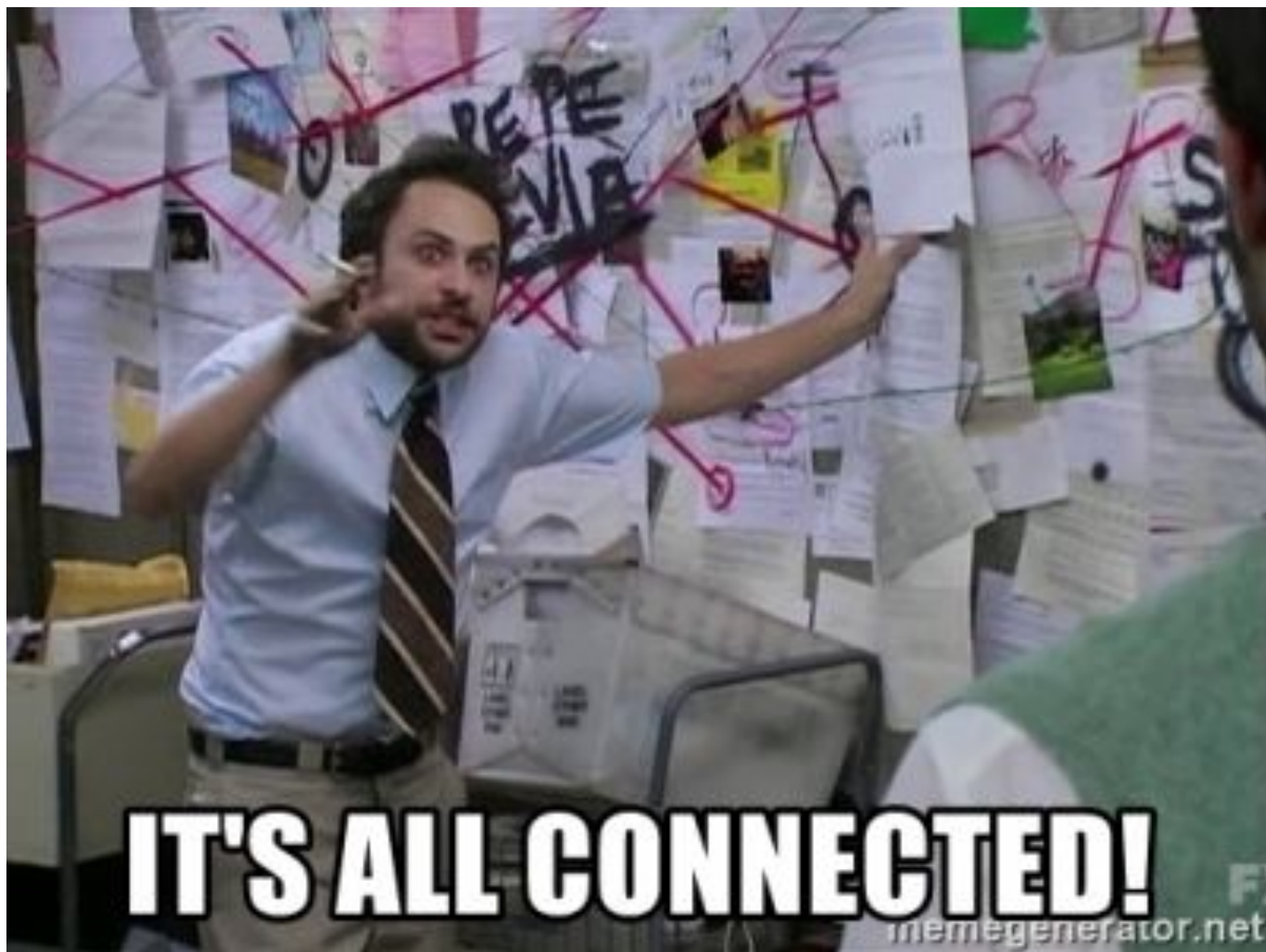
Redes neuronales profundas

Laura de la Fuente, Hernán Bocaccio

Ayudantes: Gastón Bujía, Diego Onna y Sofía Morena del Pozo

Dirección de e-mail de la materia:

datawillconfess@gmail.com



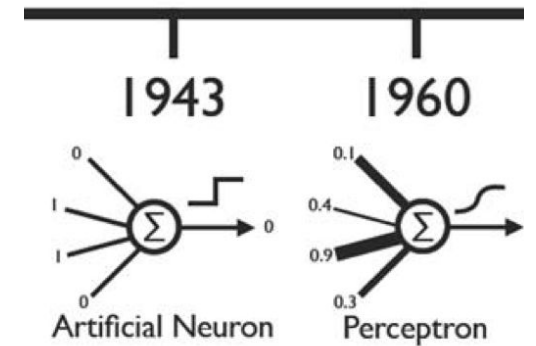
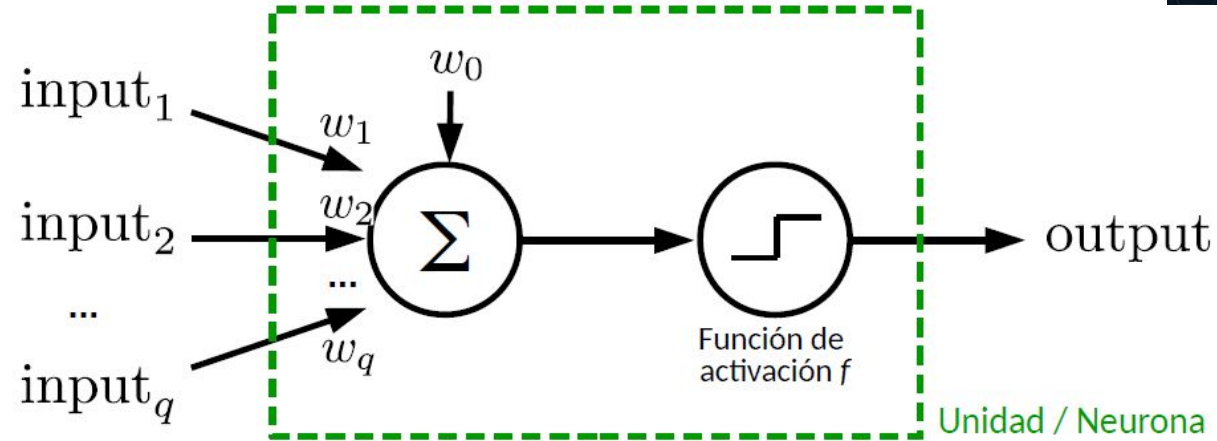
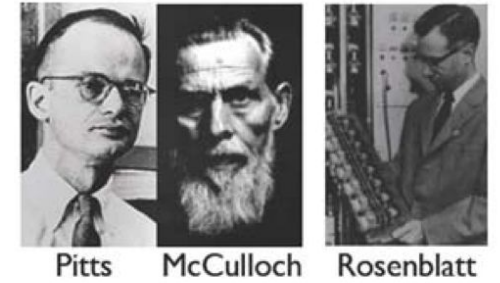
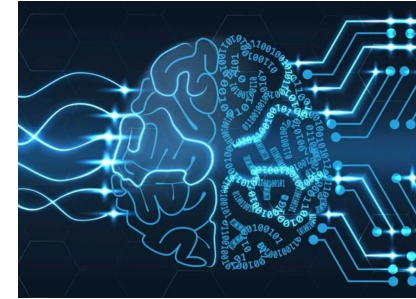
IT'S ALL CONNECTED!

Itinerario

- Perceptrón simple como problema matemático
- Perceptrón multi-capas, redes feed-forward
- Redes profundas
- Funciones de activación
- Back-propagation
- Métodos de optimización
- Batch, mini batch y epochs
- Cantidad de parámetros
- Control de varianza: regularización, dropout, etc

Perceptrón simple

Neurona artificial



Es un modelo lineal generalizado

$$y = f(x^T \cdot w) = f\left(\sum_j x_j w_j\right)$$

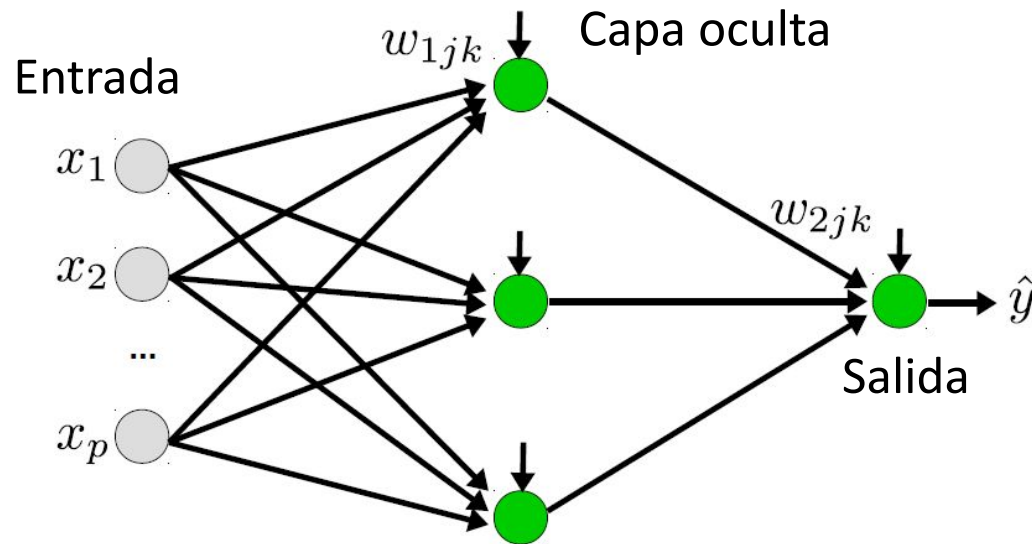
Se puede resolver con álgebra de matrices

$$f(\text{Input} \cdot W) = \text{Output}$$

$1 \times q$ $q \times 1$ 1×1

Redes feed-forward

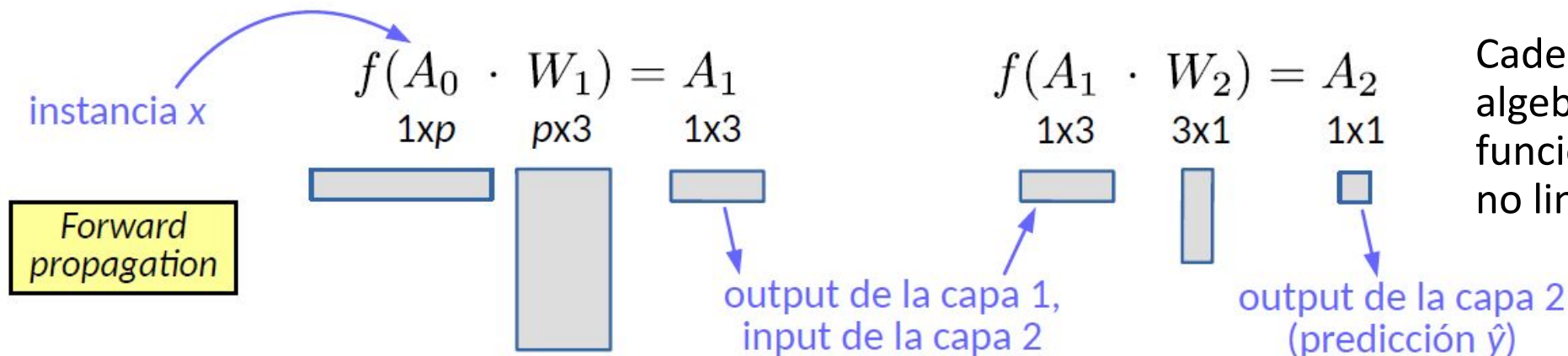
Perceptrón multi-capa



Se las llama **feed-forward** porque la información **fluye hacia adelante**

Se las llama **fully-connected** cuando cada neurona se **conecta** con **todas** las neuronas de la siguiente capa

Si tiene **más de una capa oculta** se habla de **redes profundas**

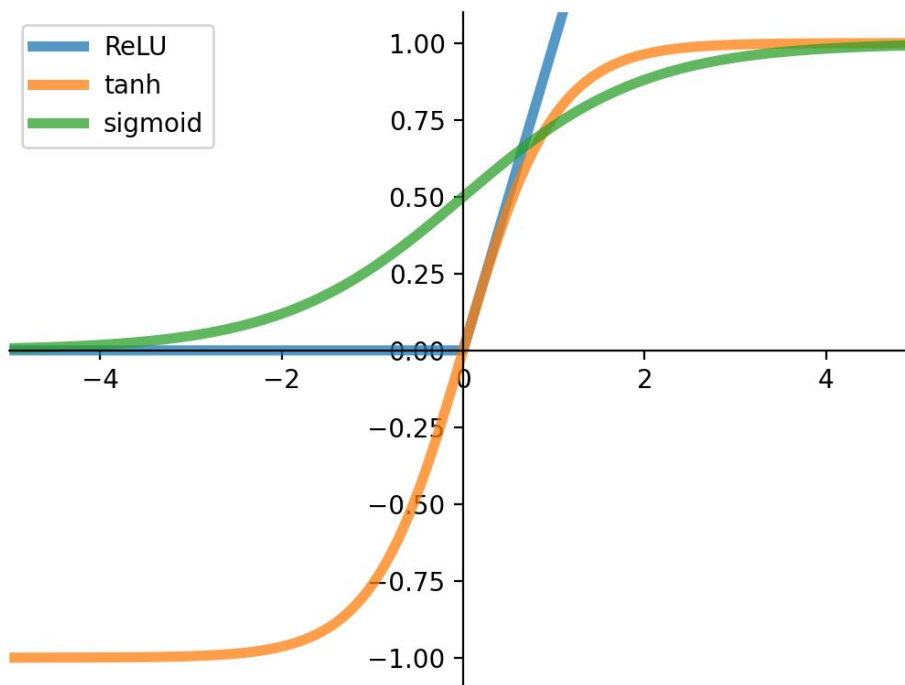


Cadena de problemas algebraicos con funciones de activación no lineal en el medio

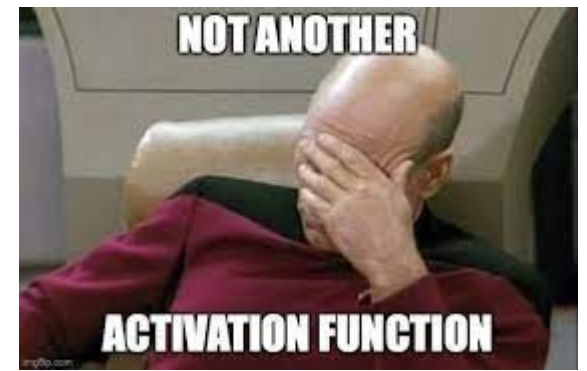
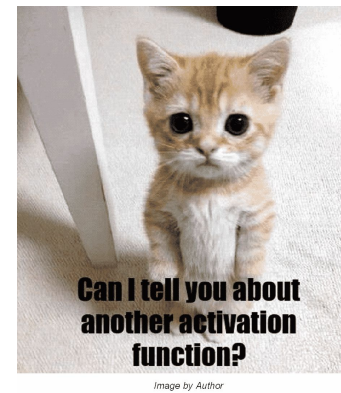
Funciones de activación

Modelo lineal generalizado

$$y(\mathbf{x}, \mathbf{w}) = f \left(w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x}) \right)$$



f es la función de activación



Tanh






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

Sigmoid

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

ReLU

$$r(a) = \max(0, a)$$

	Linear
	Sigmoid
	Tanh
	ReLU
	LeakyReLU

Redes feed-forward

Son aproximadores universales de funciones

Approximation by superpositions of a sigmoidal function

[G Cybenko](#) - Mathematics of control, signals and systems, 1989 - Springer

In this paper we demonstrate that finite linear combinations of compositions of a fixed, univariate function and a set of affine functionals can uniformly approximate any continuous ...

☆ Guardar  Citar Citado por 17859 Artículos relacionados Las 22 versiones

Approximation capabilities of multilayer feedforward networks

[K Hornik](#) - Neural networks, 1991 - Elsevier

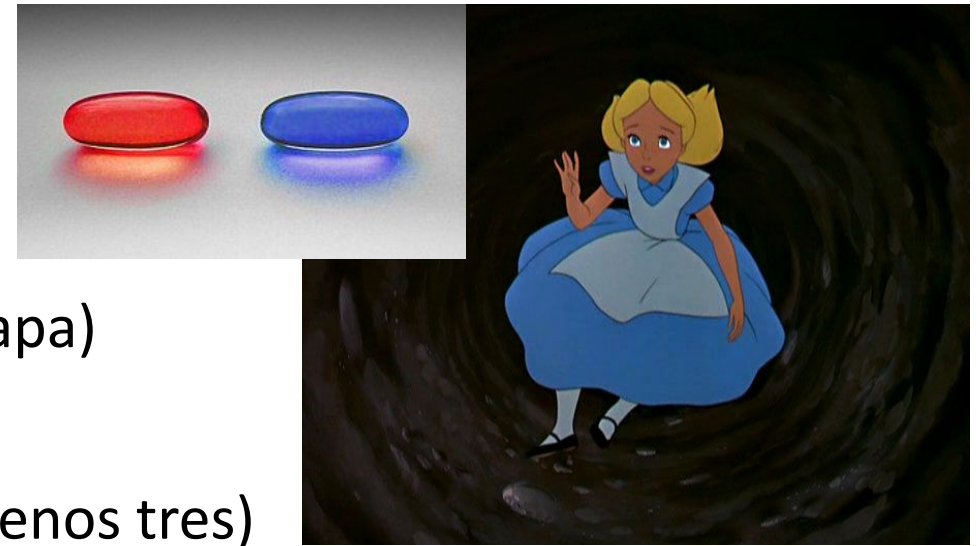
We show that standard multilayer feedforward networks with as few as a single hidden layer and arbitrary bounded and nonconstant activation function are universal approximators with ...

☆ Guardar  Citar Citado por 7183 Artículos relacionados Las 11 versiones

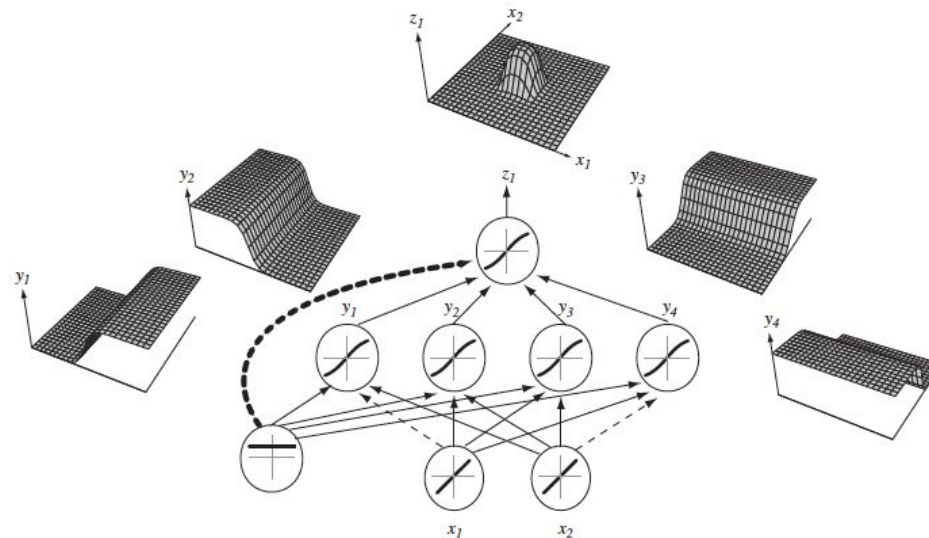
Existe una red neuronal feed-forward de una sola capa oculta, que puede aproximar cualquier función continua en un subconjunto cerrado y acotado de \mathbb{R}^n

No nos dice la cantidad de neuronas ocultas que debe tener, ni cómo obtener los pesos de dicha red neuronal

¿Qué tan profundo?



- No es un concepto nuevo (Perceptrón Multicapa)
- Procesamiento no lineal
- Múltiples capas de unidades no lineales (al menos tres)
- Énfasis en la función de cada capa como extractores de características (la información de cada capa y transformada por la siguiente)
- Niveles jerárquicos de representación con grados crecientes de abstracción



Back-propagation

Cómo actualizo los pesos?

Rumelhart, Hinton, and Williams 1986

Necesitamos un algoritmo eficiente que nos permita adaptar todos los pesos de una red multicapa, no sólo los de la capa de salida

Aprender los pesos correspondientes a las neuronas de las capas ocultas equivale a aprender nuevas características (no presentes en el conjunto de entrenamiento), lo que resulta especialmente difícil porque nadie nos dice directamente qué es lo que deberíamos aprender en esas unidades ocultas

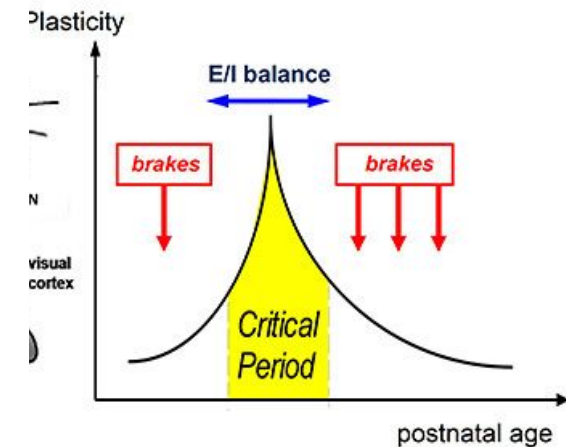
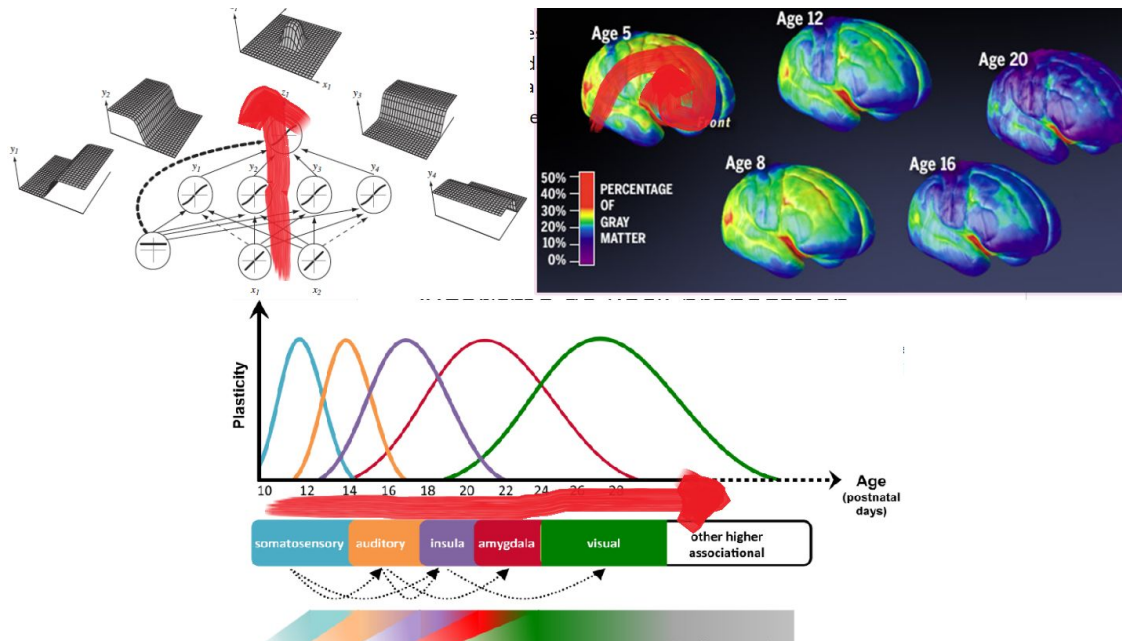
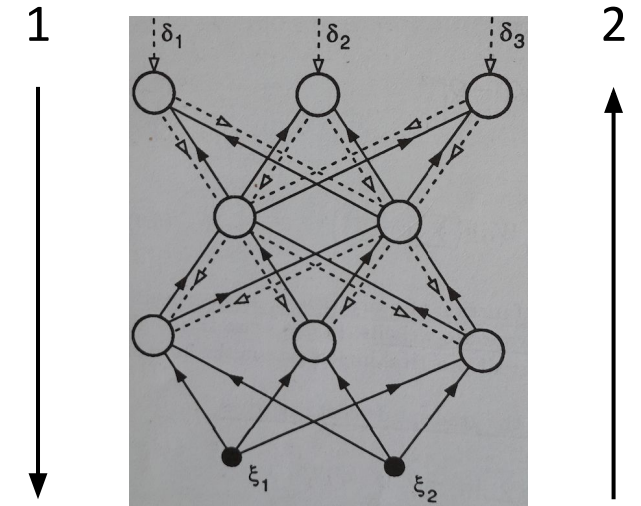
Calculo las derivadas, pero aplico regla de la cadena y voy propagando el gradiente hacia atrás de a pasos, lo cual me permite actualizar los pesos

$$w_{ijk}^{[t+1]} \leftarrow w_{ijk}^{[t]} - \eta \frac{\partial J(\theta)}{\partial w_{ijk}}$$

Algoritmo de Back-propagation

La regla de aprendizaje está basada en minimizar la función de costo $E(w)$, siguiendo la dirección en la que desciende el gradiente

1. Etapa forward: Calcula el output de la red neuronal
2. Etapa backward: Calcula el gradiente descendiente de $E(w)$ con respecto a los pesos de la red, y actualiza los pesos

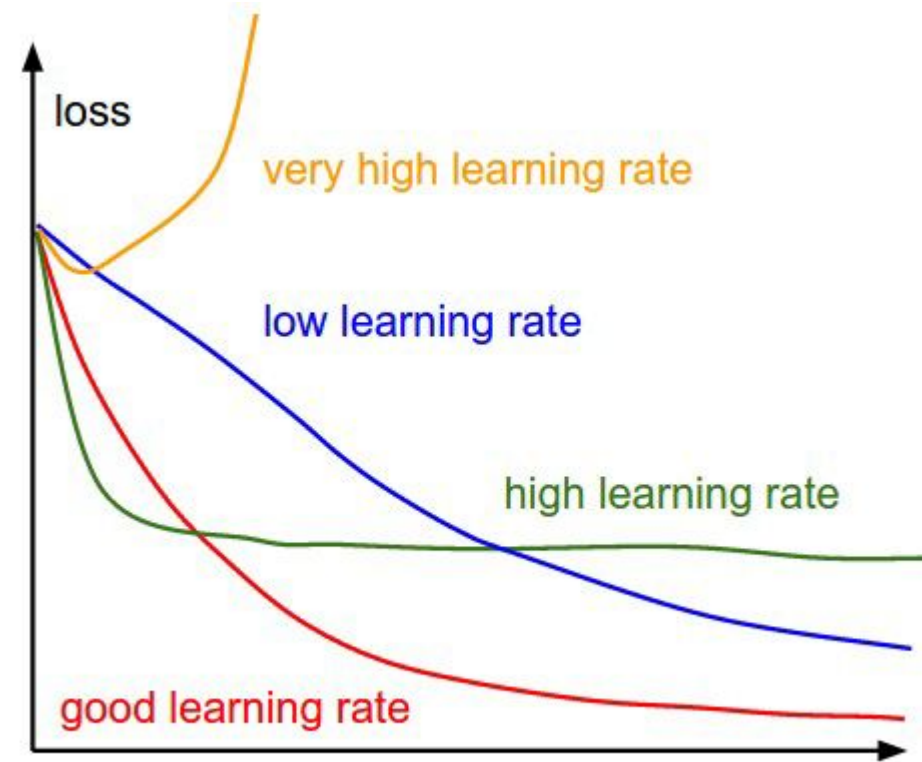


Factor de aprendizaje o Learning Rate

El factor de aprendizaje es el escalar que aparece en la regla Δw de actualización de los pesos, y equivale al tamaño del paso, en la dirección de descenso de la función de costo

Muy grande, diverge

Muy pequeño, tardamos demasiado en converger

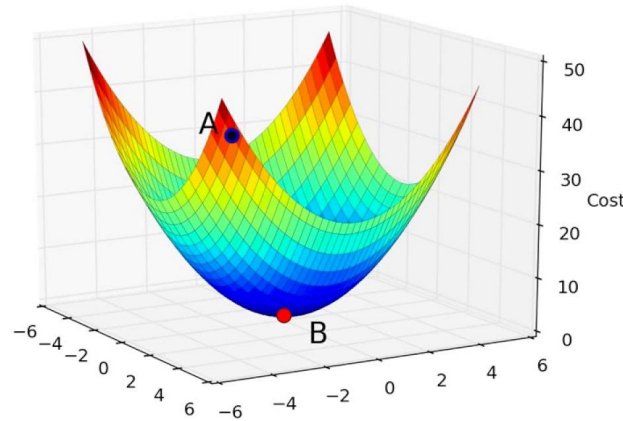


Función de costo

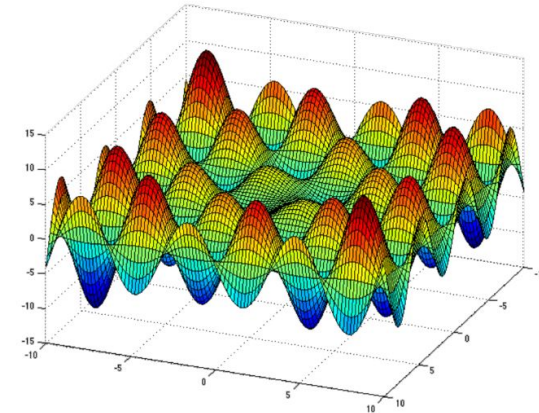
The loss surfaces of multilayer networks

[A Choromanska](#), [M Henaff](#), [M Mathieu](#)... - Artificial intelligence ..., 2015 - proceedings.mlr.press

Perceptrón
simple
Convexa



Perceptrón
multi-layer
No convexa



Gradient descent no garantiza encontrar el mínimo global en funciones no-convexas

El destino depende de la inicialización, depende de información local

Sin embargo, para redes neuronales grandes, la mayoría de los mínimos locales son similares y presentan performance similar en los dataset de test

La probabilidad de encontrar mínimos locales 'malos' decrece con el tamaño de la red

Focalizarse demasiado en la búsqueda del mínimo global en el dataset de entrenamiento no es útil en la práctica, y puede terminar en sobreajuste del modelo

Muchos mínimos locales

Al haber muchos mínimos locales, para un aprendizaje óptimo, hay dos caminos.

Estrategias:

- Mejorar el método de la búsqueda del mínimo (momentum, inicialización de pesos, entrenamiento por minibatch, etc)
- Modificar la forma de la $E(W)$ para que sea más fácil encontrar un 'buen' mínimo (Regularización, agregar neuronas a la red, agregar capas, etc)

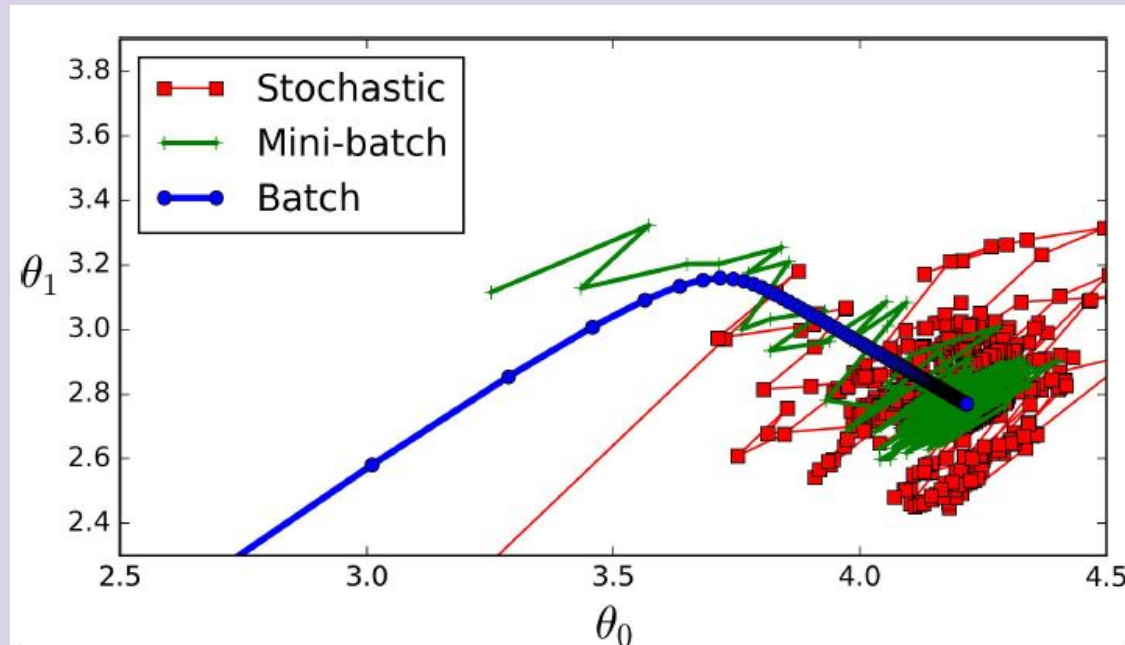


Todos los hiper parámetros que no vamos a cross-validar

Métodos de optimización

Descenso por el gradiente

Actualizo los pesos en cada iteración, pero no necesariamente con todos los datos



“Batch” Gradient Descent:

Uso todos los datos de entrenamiento para calcular la función costo y el gradiente

“Stochastic” Gradient Descent:

Uso una única instancia de los datos, al azar, para calcular la función costo y el gradiente

“Mini-batch” Gradient Descent:

Uso un subconjunto de datos de entrenamiento para calcular la función costo y el gradiente

Existen muchos **otros métodos** para actualizar los pesos, todos son **variaciones del descenso por gradiente** (ej. Momentum, Adam, Adagrad, RMSProp, etc)

Métodos de optimización

Momentum

La idea es incorporar inercia al término de actualización de los pesos, esto quiere decir que dependa del valor de actualización de la iteración anterior

Se busca acelerar el proceso de convergencia y ayudar a superar mínimos locales (sobre todo en SGD)

$$v_t = \gamma v_{t-1} + \eta \nabla_{\theta} J(\theta)$$
$$\theta = \theta - v_t$$



Image 2: SGD without momentum

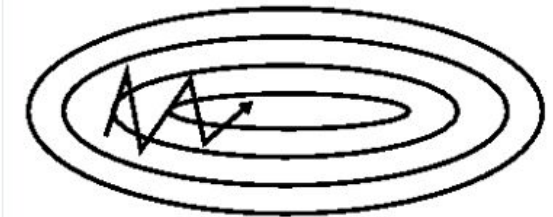


Image 3: SGD with momentum

Adaptativos (RMSProp, Adadelata, Adagrad)

Los métodos adaptativos, en lugar de usar un Learning Rate global para todos los parámetros, usan un LR escalado por cada parámetro

Métodos de optimización

Adam

Además de la inercia, el método ajusta el Learning Rate para cada parámetro teniendo en cuenta el cuadrado del gradiente correspondiente a ese parámetro

Actualización

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

m y v: medias y varianzas de los parámetros

Hoy en día es el método de optimización más usado

[Adam: A method for stochastic optimization](#)

[DP Kingma](#), [J Ba](#) - arXiv preprint arXiv:1412.6980, 2014 - arxiv.org

We introduce Adam, an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. The method is ...

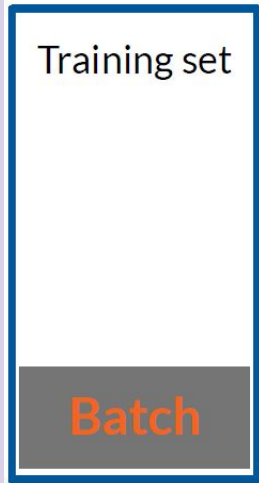
☆ Guardar  Citar Citado por 144486 Artículos relacionados Las 21 versiones 

Inicialización de pesos

Inicialización dependiente de función de activación

Activation function	Uniform distribution $[-r, r]$	Normal distribution
Logistic	$r = \sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$	$\sigma = \sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$
Hyperbolic tangent	$r = 4\sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$	$\sigma = 4\sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$
ReLU (and its variants)	$r = \sqrt{2}\sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$	$\sigma = \sqrt{2}\sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$

Batch, mini batch y epochs

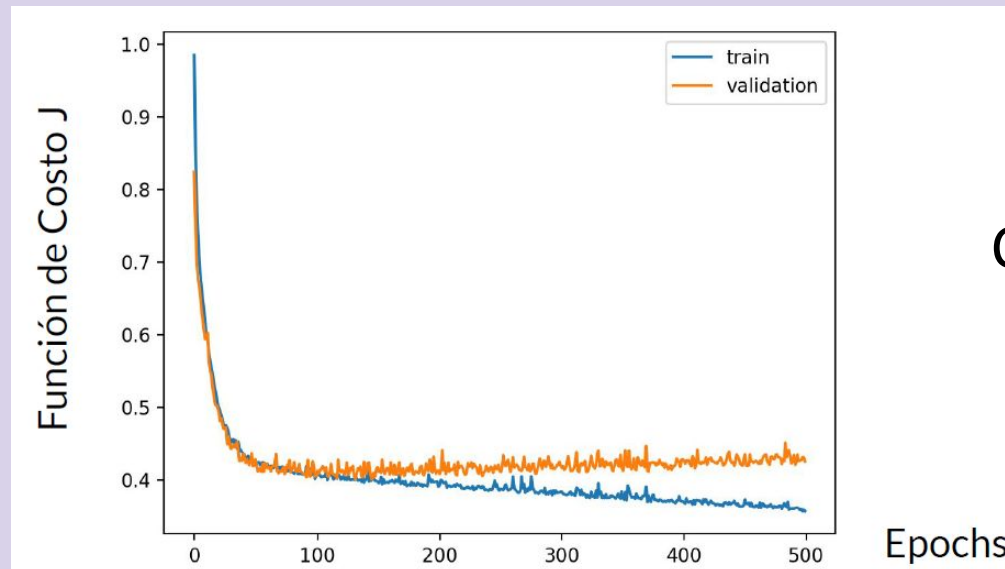


Una **Iteración** es una pasada de forward y back en la que calculo la función costo, su gradiente, y actualizo los pesos de la red

Una **Epoch** es la cantidad de veces que pasamos el training set completo por la red

Cuando el batch tiene menos instancias que el total de instancias de entrenamiento, entonces vamos a necesitar **varias iteraciones para completar cada epoch (mini batch)**

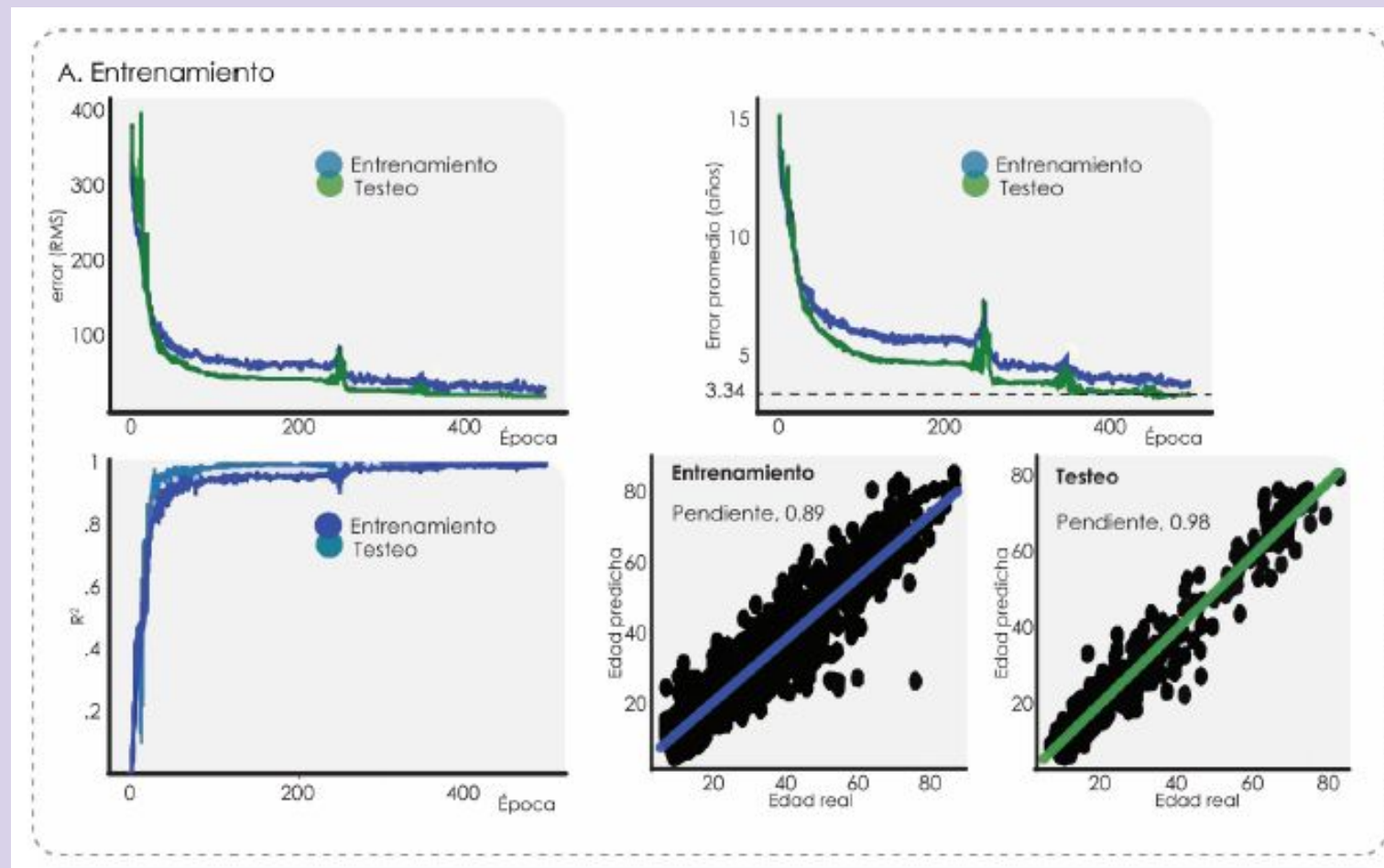
En general se precisan **varios Epochs para entrenar la red**



Curva de entrenamiento

Learning rate variables

En estos casos se habla de un cierto número de ciclos con un número de épocas en subida y bajada (puede ser exponencial, lineal, etc)



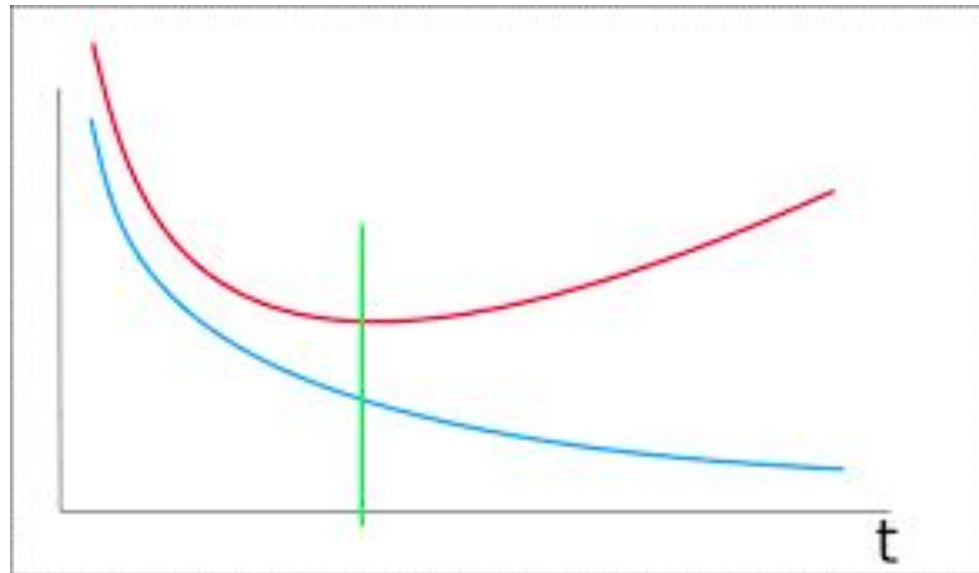
Entrenamiento: Procedimiento general

Al final de cada época, medimos los 'errores' de los conjuntos de entrenamiento y validación. Mientras estos errores sigan disminuyendo, continuamos el entrenamiento

Cuando el 'error' del conjunto de validación comienza a ascender, detenemos el entrenamiento (early-stopping) para evitar un overfitting, y medimos el error del conjunto de testing

Si los 'errores' de los conjuntos de training y testing, son lo suficientemente bajos, terminamos

Sino, habrá que conseguir más datos de entrada, o modificar la estructura de la red, ya que puede estar sucediendo que la red no sea lo suficientemente compleja como para modelar el problema



‘Error’ = $f(\text{Capa de salida})$

Problem	Output Layer		
	Size	Activation	Error
Regression	N	$f(x) = x$	MSE RMSE
Binary Classification	1	$f(x) = \text{sigmoide}$	Cross-entropy
Multi-class classification	K	$f(x) = \text{softmax}$	Multiclass Cross-entropy

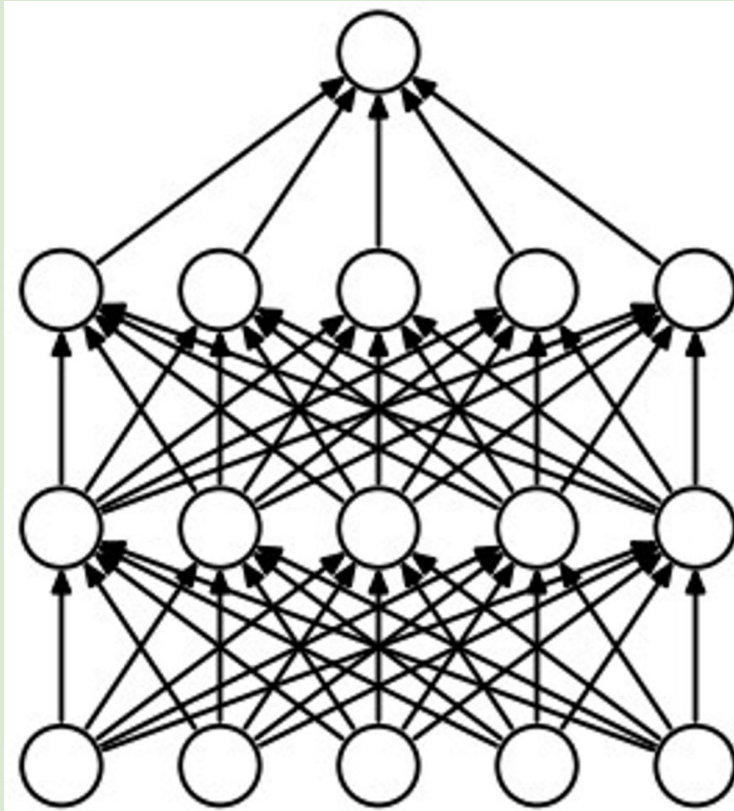
Regularización

En razón de la gran cantidad de parámetros, las redes neuronales tienden a sobreajustar los datos de entrenamiento.

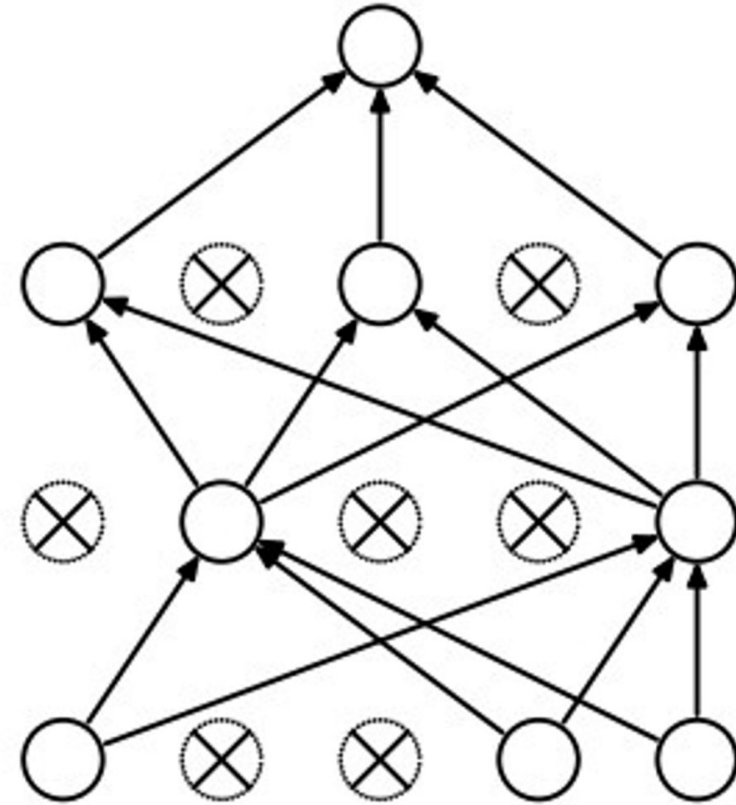
Se puede usar una serie de medidas para evitar esto:

- Regularización L2, llamada weight-decay
- Early stopping
- Weight-sharing (grupos de unidades comparten los pesos)
- Dropout
- ...

Dropout

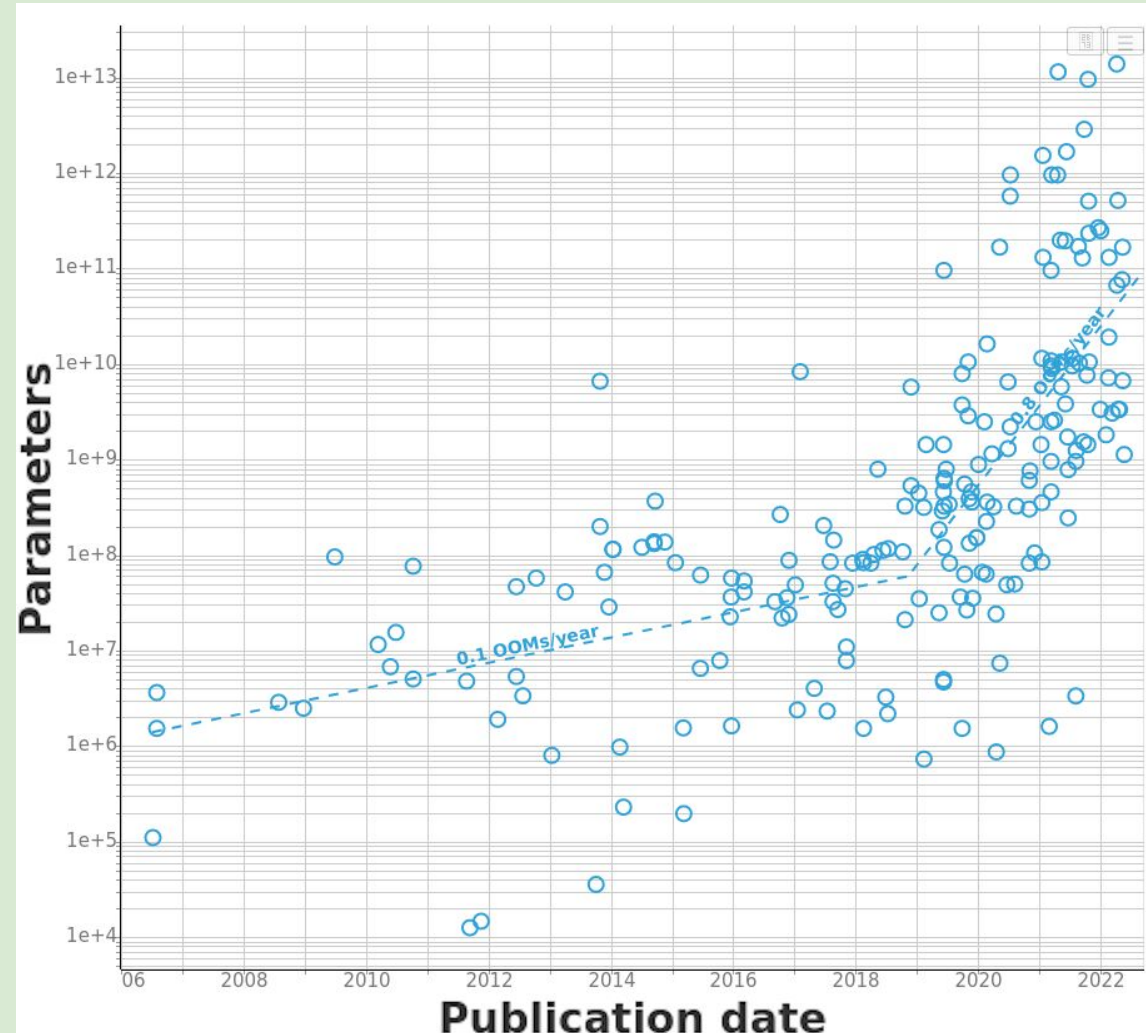


(a) Standard Neural Net



(b) After applying dropout.

Número de neuronas o capas = Cantidad de parámetros



Cantidad de parámetros

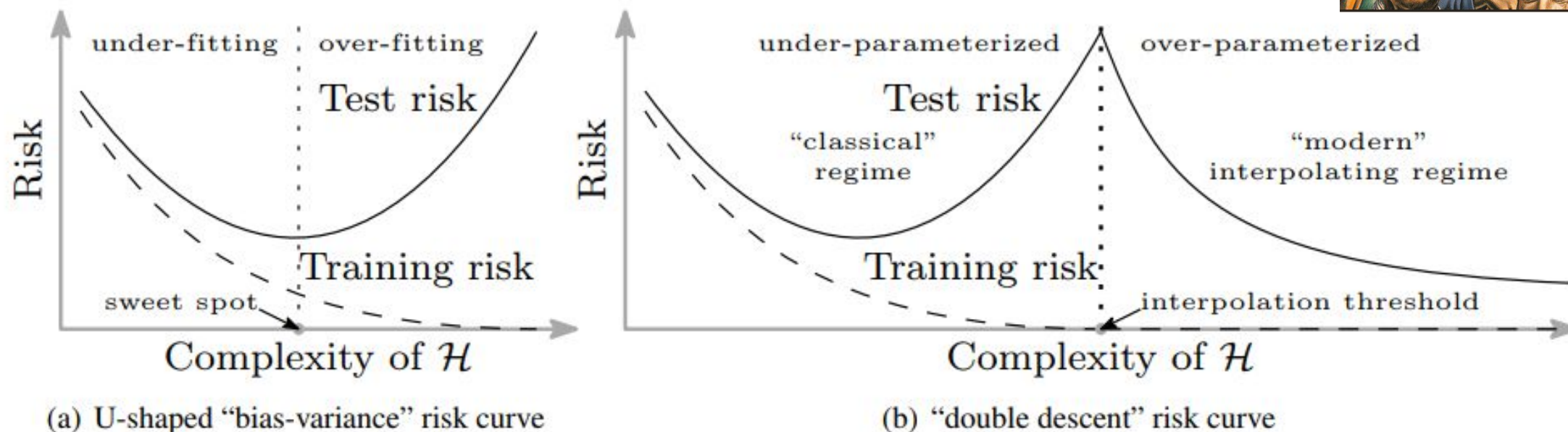
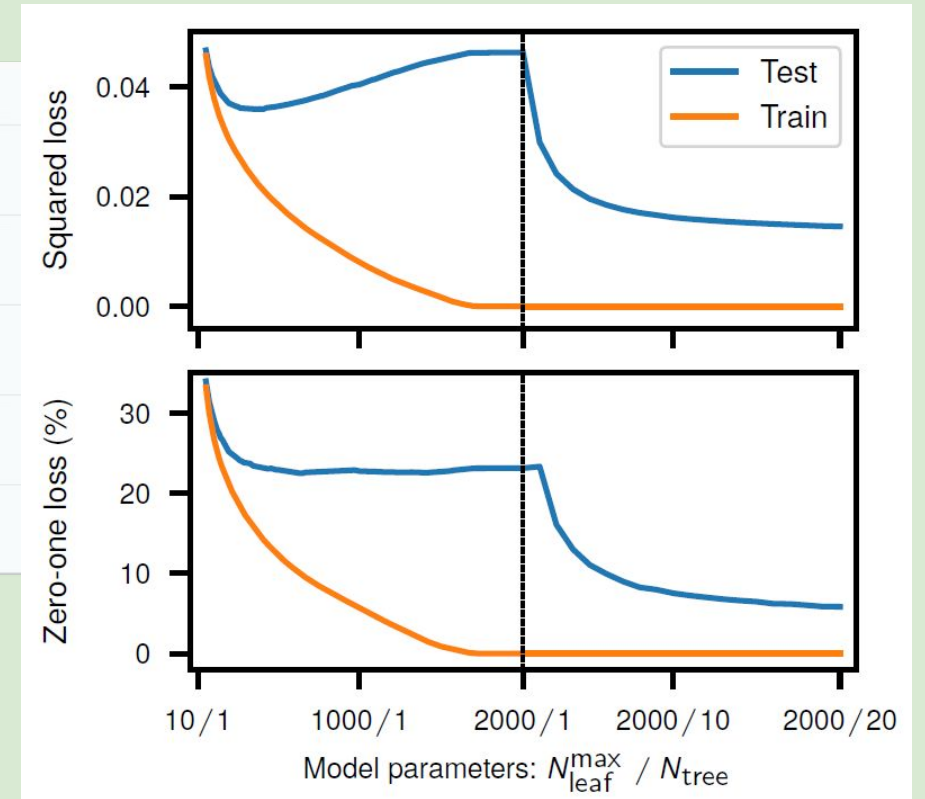
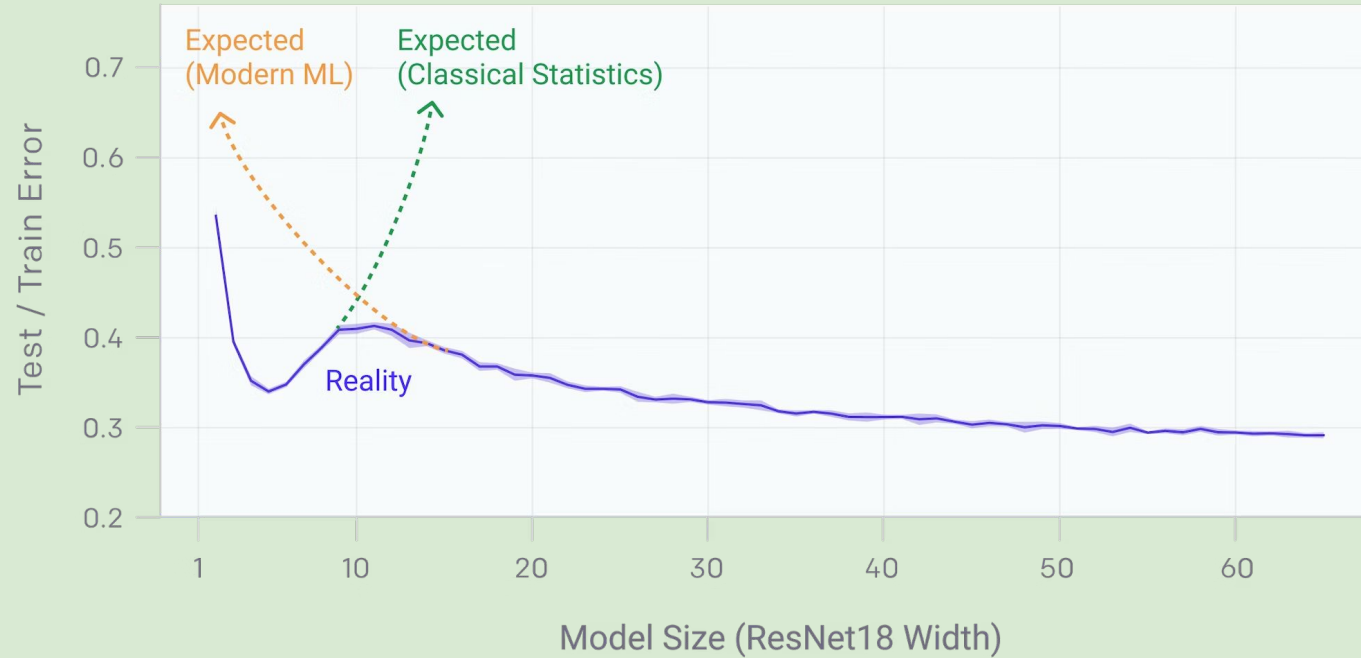
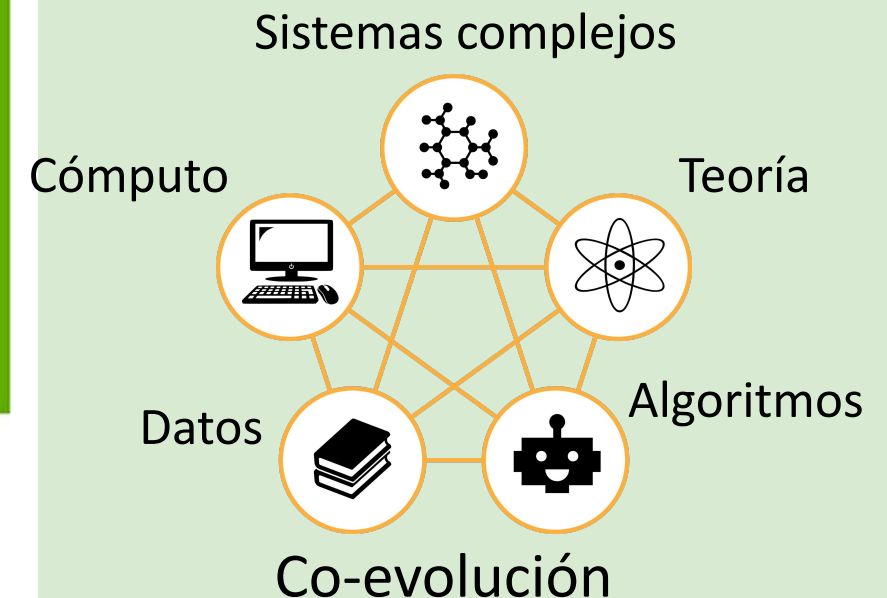
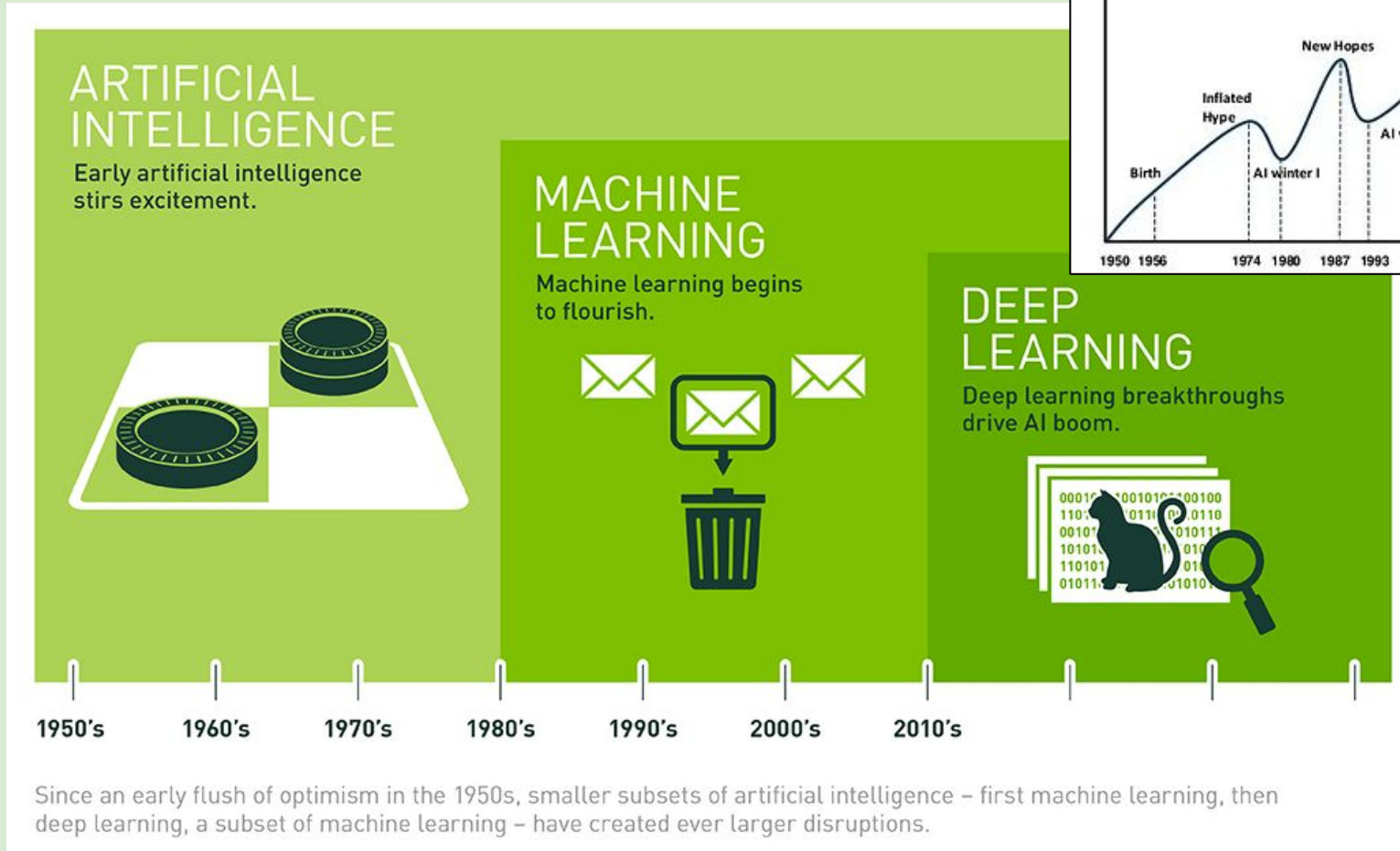


Figure 1: Curves for training risk (dashed line) and test risk (solid line). (a) The classical *U-shaped risk curve* arising from the bias-variance trade-off. (b) The *double descent risk curve*, which incorporates the U-shaped risk curve (i.e., the “classical” regime) together with the observed behavior from using high complexity function classes (i.e., the “modern” interpolating regime), separated by the interpolation threshold. The predictors to the right of the interpolation threshold have zero training risk.

Cantidad de parámetros

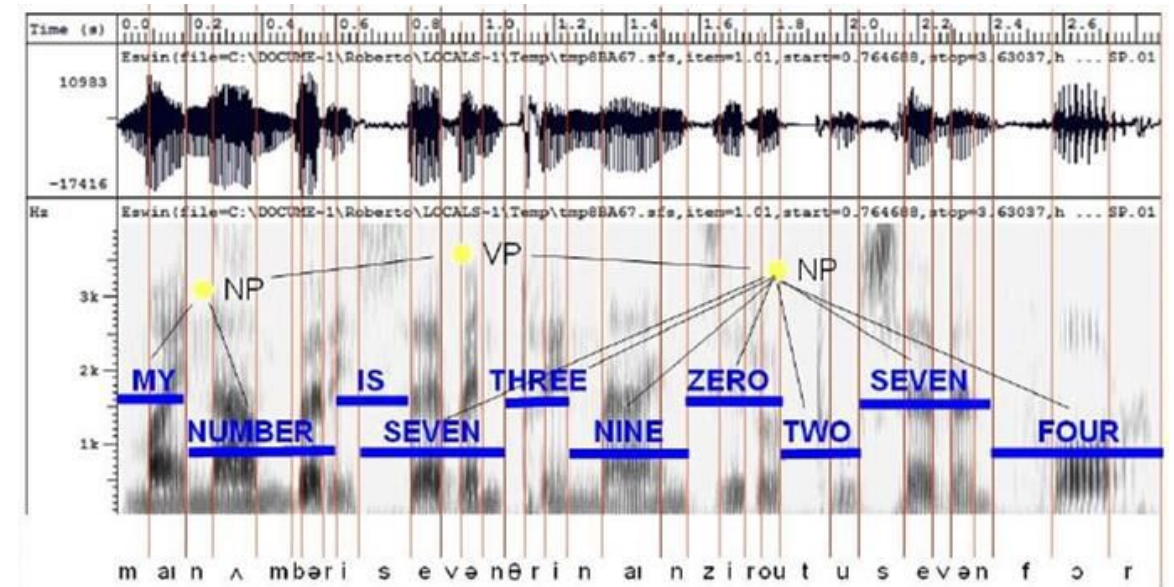
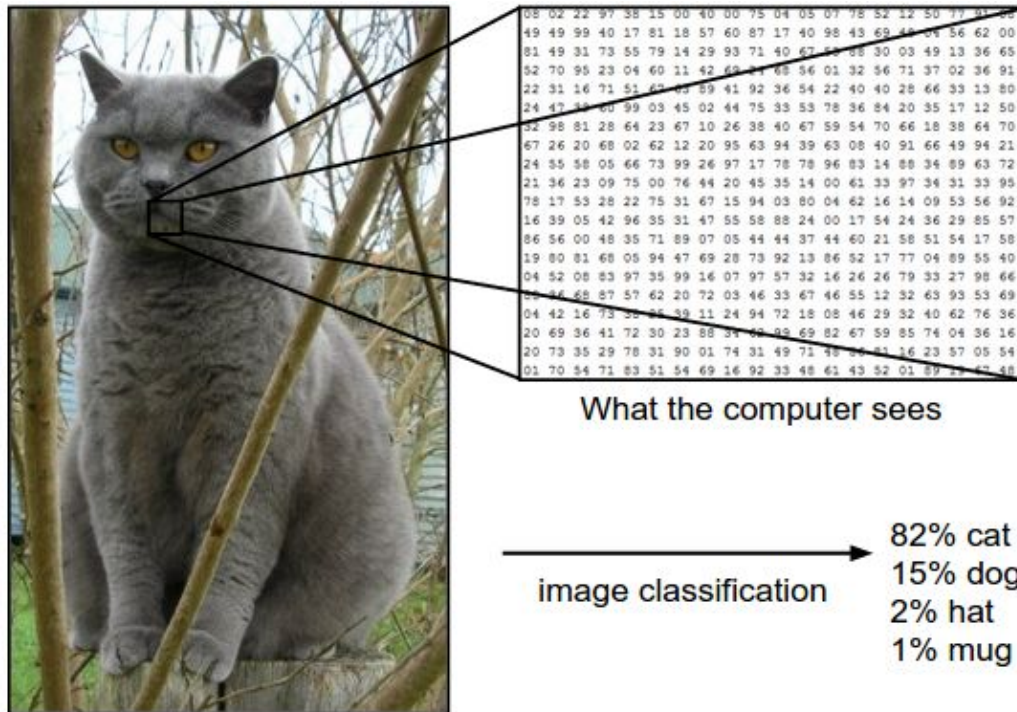


Explosión del Aprendizaje Profundo



Explosión del Aprendizaje Profundo

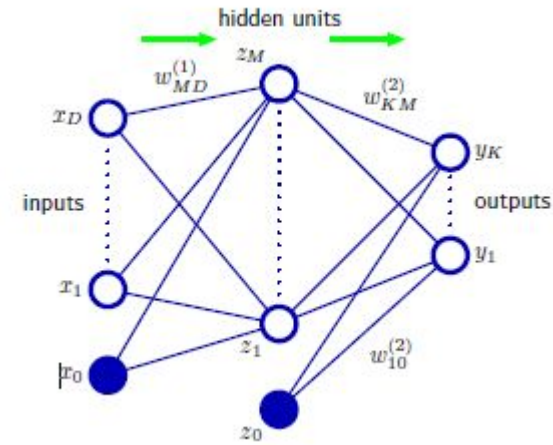
Relevancia para datos no tabulados



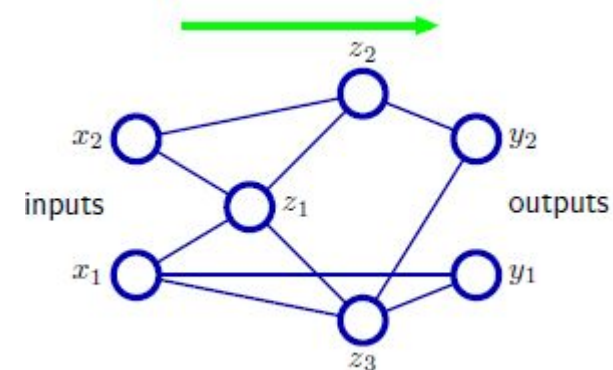
Imágenes, series temporales, procesamiento de lenguaje natural, etc

La arquitectura puede generalizarse

- incluyendo más capas ocultas



- redes ralas (que no tienen todas las conexiones)



- conexiones que salten capas

Aaaa-los-co-labs...

