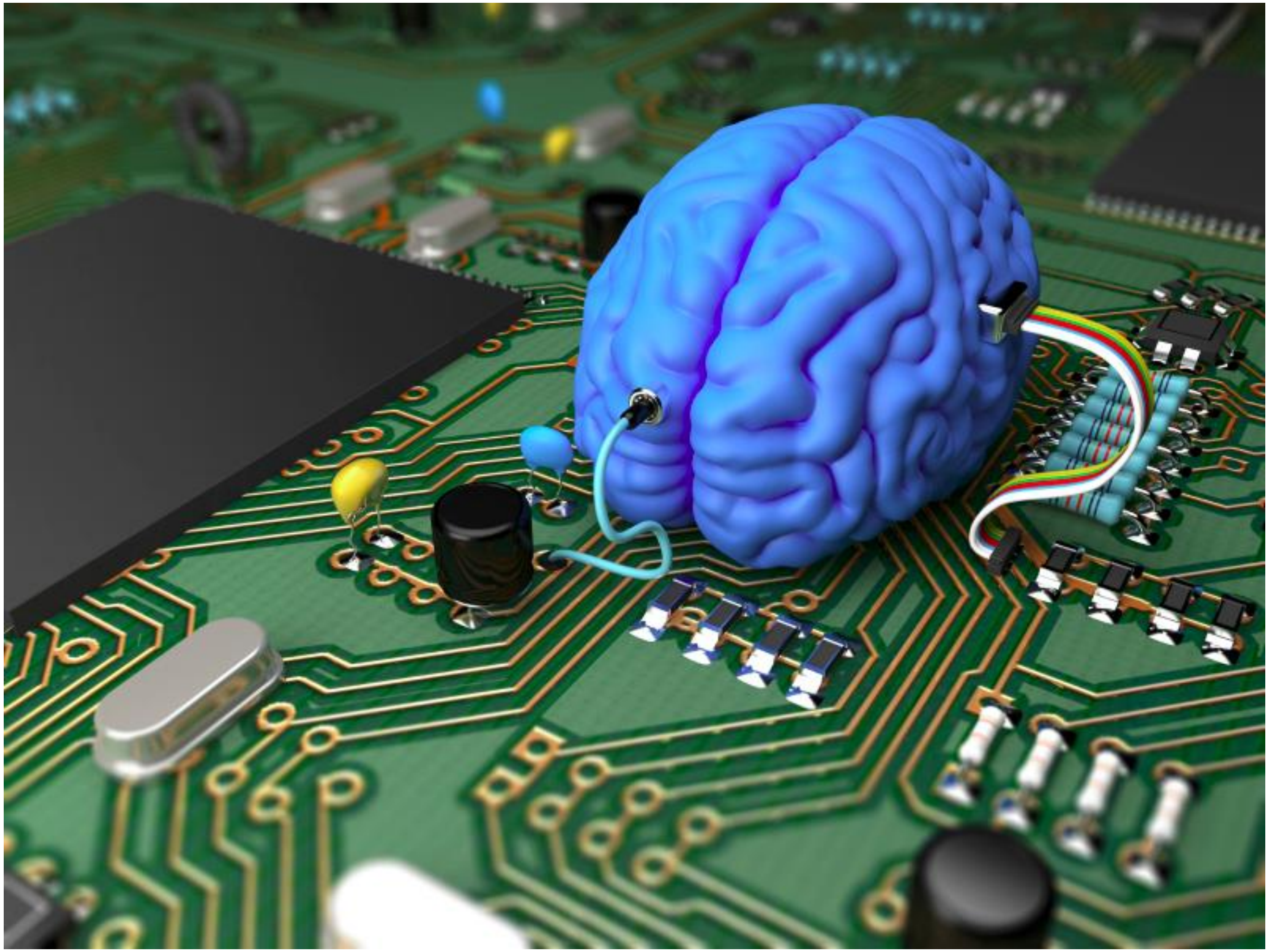
Machine learning I : Deep Learning

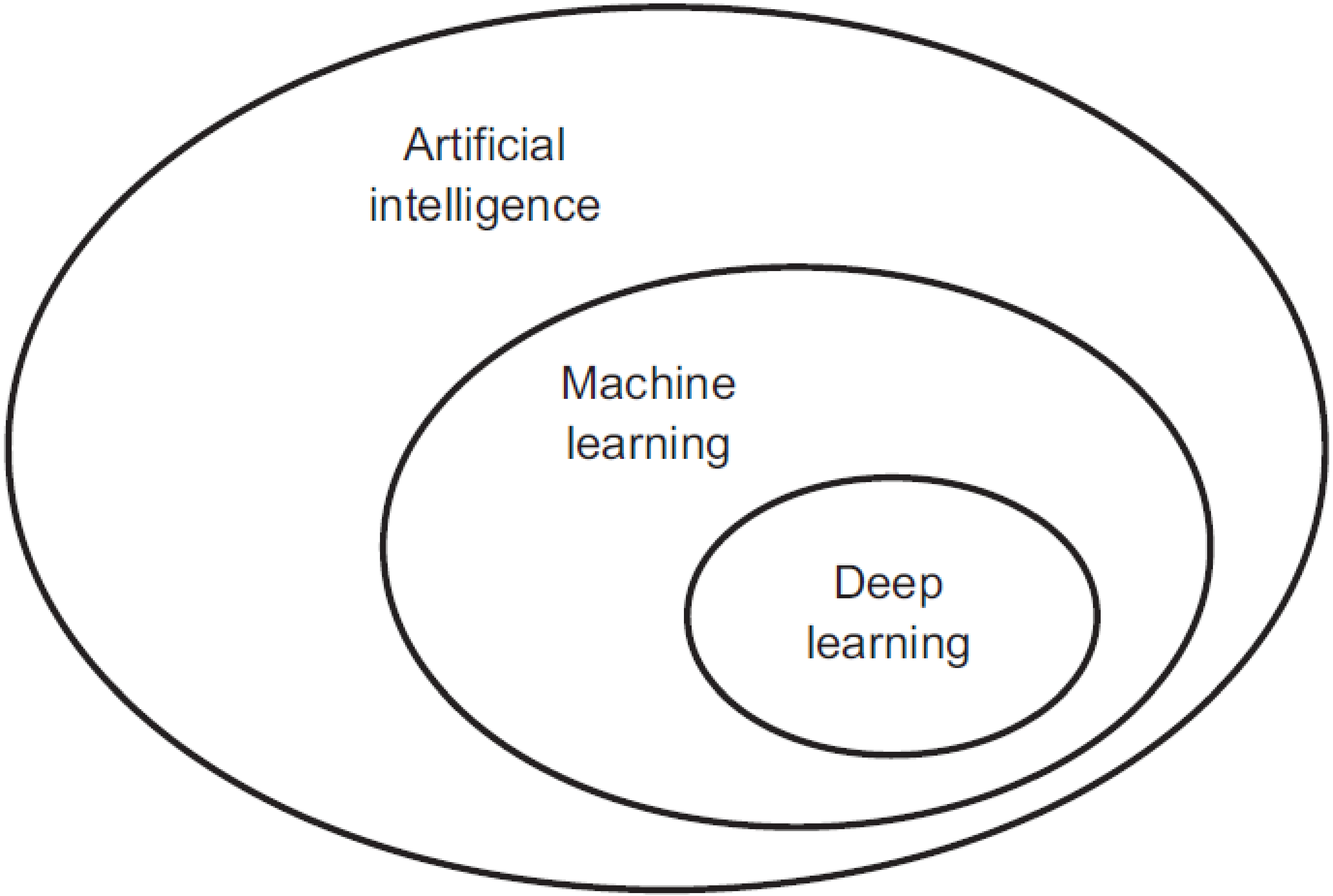


Master Universitario Oficial **Data Science**
UC Universidad de Cantabria
UIMP Universidad Internacional Menéndez Pelayo
con el apoyo del CSIC

Lara Lloret Iglesias

Santander, Febrero 2019

Introducción



Repaso: Machine Learning

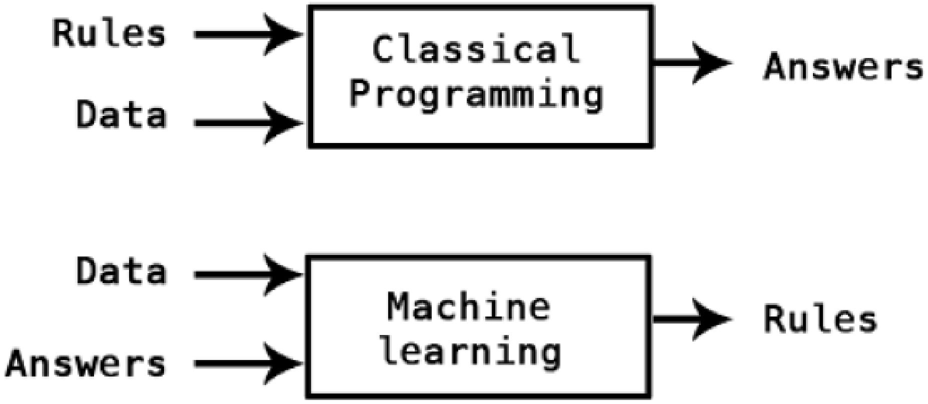
El Machine learning surge de una serie de preguntas:

¿Puede un ordenador ir más allá de "lo que sabemos como ordenarle"?
¿Puede de verdad aprender una tarea concreta "a su manera"?

¿Puede sorprendernos?

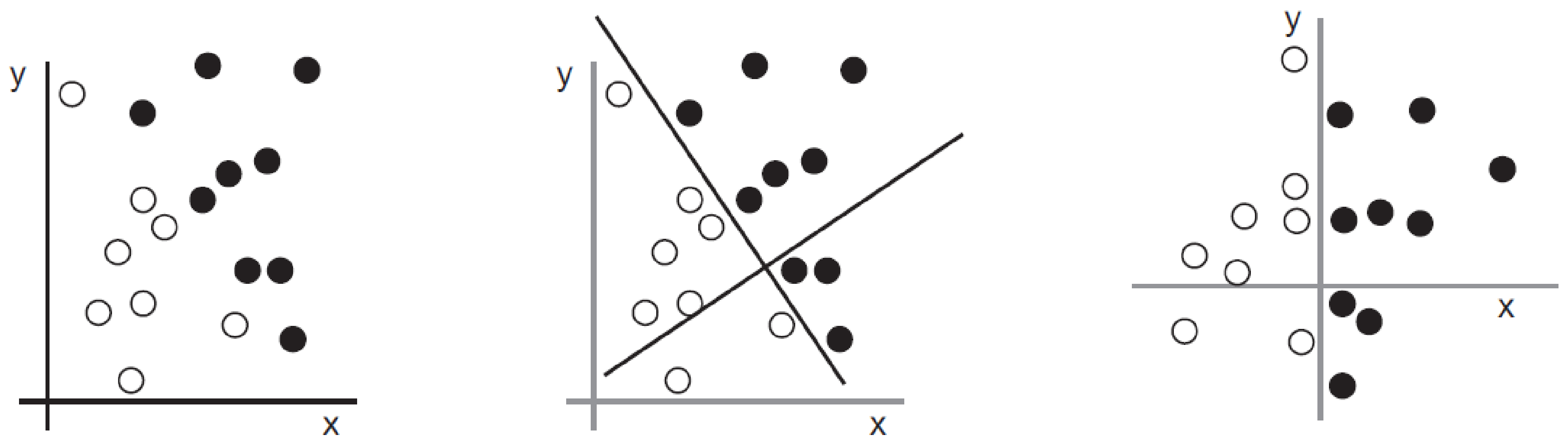
En lugar de darle nosotros ciertas reglas codificadas…

¿Puede aprender automaticamente mirando a los datos?



Repaso: representaciones

- La base de Machine Learning (y Deep Learning) es transformar datos de una manera significativa que nos permita resolver el problema
- ¿Qué es una representación? ➡ Otra manera de mirar a los datos
- Un problema muy difícil en una representación se puede volver trivial en otra



Repaso: aprendizaje

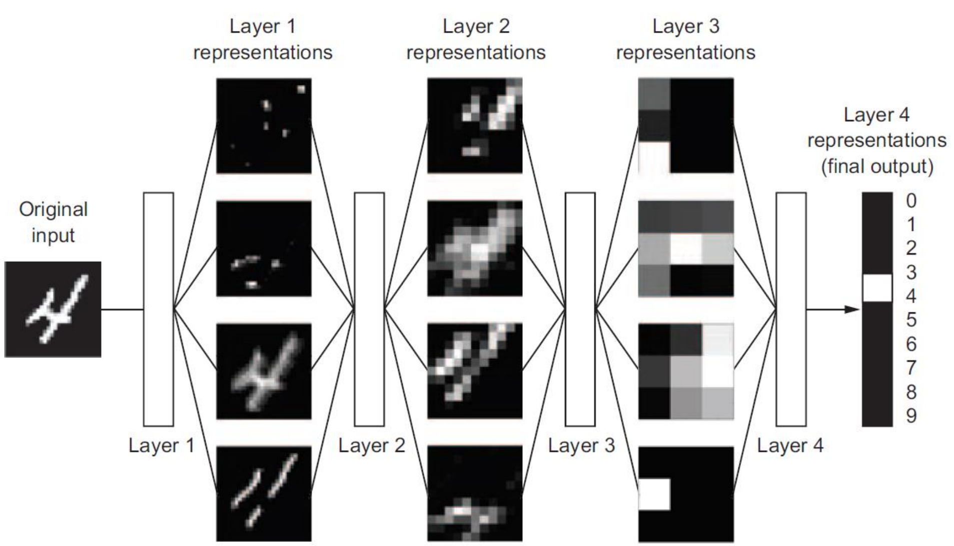
En el ejemplo anterior hemos definido el cambio de coordenadas “a mano”, pero… y si en su lugar lo hiciéramos:

- De manera sistemática
- Probando con varios cambios de coordenadas diferentes
- Usando como feedback el porcentaje de puntos que están siendo correctamente clasificados

El aprendizaje en Machine Learning/Deep Learning describe un sistema de búsqueda automática de representaciones mejores para el problema en cuestión.

Repaso: ¿Qué significa “profundo”?

- Deep Learning es un enfoque diferente del aprendizaje de nuevas representaciones.
- Lo “profundo” no hace referencia a un entendimiento más profundo de los datos sino a aprender **capas sucesivas de representaciones cada vez más significativas.**
- El número de estas capas es lo que se conoce como la **profundidad del modelo**.



¿Por qué es diferente el Deep Learning?

- El Deep Learning ofrece un mayor rendimiento en general.
- Con Deep Learning la resolución de problemas resulta mucho más sencilla porque **automatiza** lo que antes era el problema crucial en machine-learning: **la ingeniería de características** (feature engineering).
- **Las técnicas anteriores** de Machine Learning transformaban los **datos solo en una o dos representaciones diferentes**. Normalmente a través de transformaciones simples como proyecciones no lineales (SVMs) o Árboles de Decisión.
- Esto implicaba que para sacar el máximo potencial de un conjunto de datos hubiera que **extraer previamente a mano representaciones/características** de los datos que ayudaran a clasificar.
- Con Deep Learning se aprenden las características "al vuelo" sin necesidad de hacer transformaciones previas a mano
- Además estas características son aprendidas al mismo tiempo: **todos los parámetros son actualizados a la vez**.

¿Por qué ahora?

Las ideas principales de Deep Learning ya se entendían en 1989.

Entonces…**¿por qué ahora?**

Los motivos principales son:

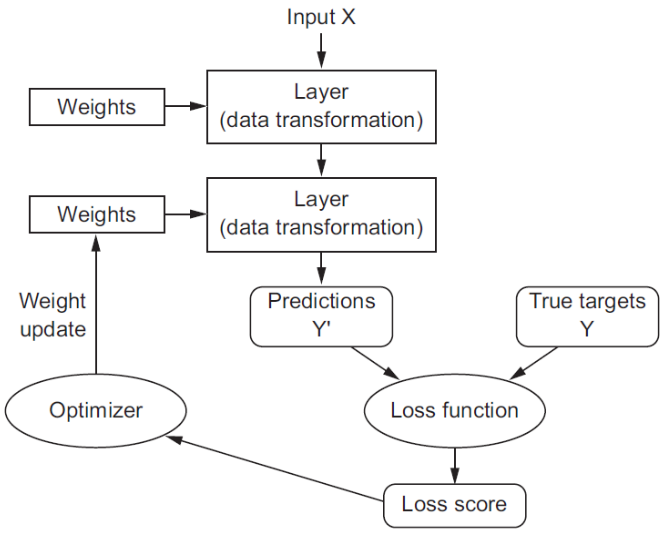
- **Hardware:**
  - Entre 1990 y 2010 las CPUs se hicieron aproximadamente 5000 veces más rápidas.
  - Desarrollo de GPUs: una GPU puede sustituir a clusters masivos de CPUs en tareas altamente paralelizables (i.e multiplicación de matrices).
- **Datasets y benchmarks:**
  - Además de las enormes mejoras en almacenamiento de datos, **el auténtico cambio vino de la mano de internet**, haciendo posible recolectar y distribuir enormes dataset de una manera sencilla y eficiente.
- **Avances algorítmicos*:**
  - Mejores funciones de activación
  - Mejores esquemas de inicialización de los pesos
  - Mejores esquemas de optimización, como **RMSProp** y **Adam**

* Estos avances no fueron posibles hasta que no se produjeron los avances en el hardware y en los datasets accesibles.

¿Por qué ahora?

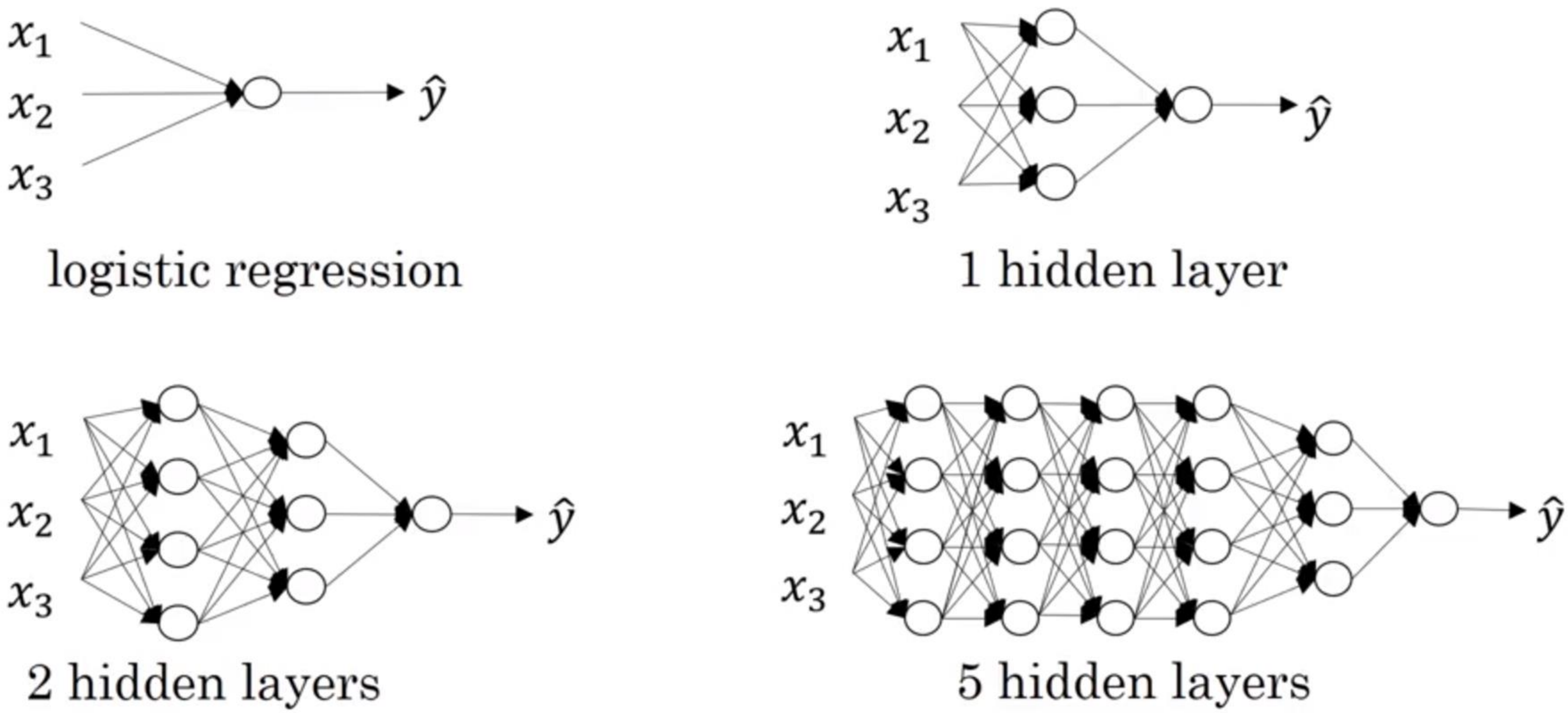
Redes Neuronales

- En Deep Learning estas "representaciones en capas" casi siempre suelen aprenderse a través de unos modelos llamados **Redes Neuronales** (ya vistas en la primera parte de esta asignatura)

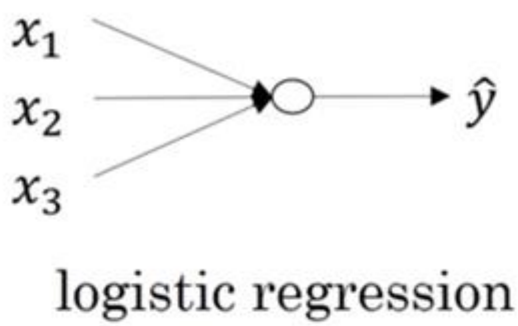


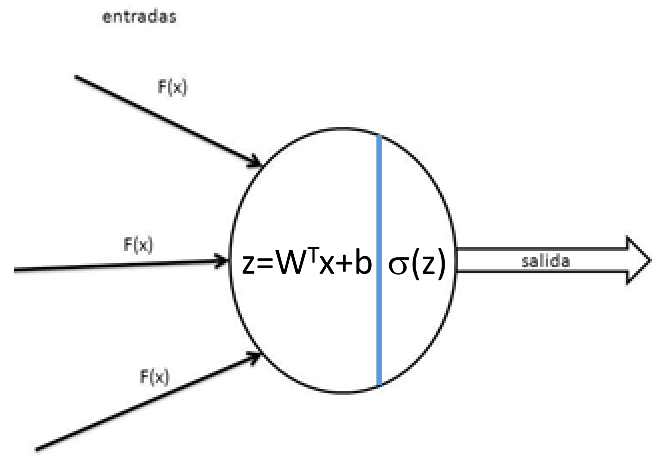
- El estado del arte en visión artificial incluye un modelo de Redes Neuronales llamadas **Redes Convolucionales**

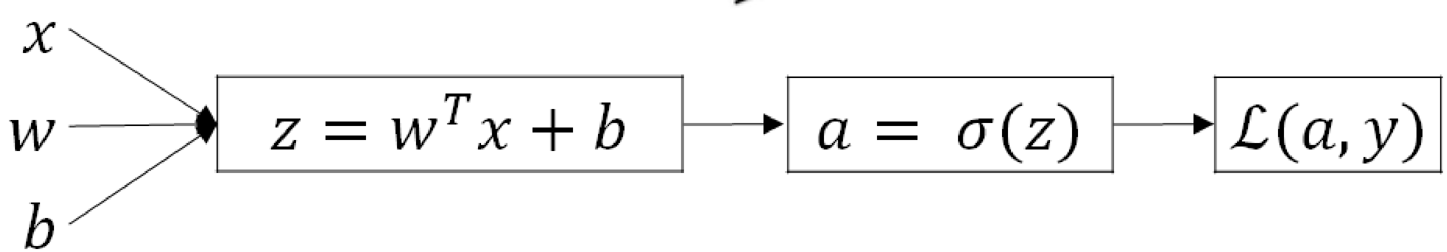
Redes Neuronales Deep



Logistic regression: Gradient descent







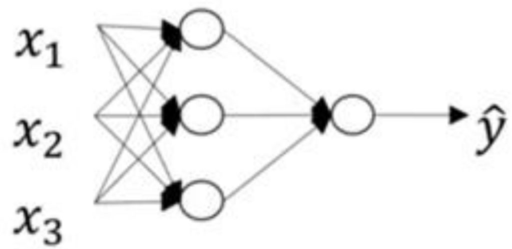
$$\frac{dL}{dw} = dw = \frac{dL}{da}\frac{da}{dz}\frac{dz}{dw}$$

$$\frac{dL}{db} = db = \frac{dL}{da}\frac{da}{dz}\frac{dz}{db}$$

$$W = W - \alpha\, dW$$

$$b = b - \alpha\, db$$

Logistic regression: Gradient descent



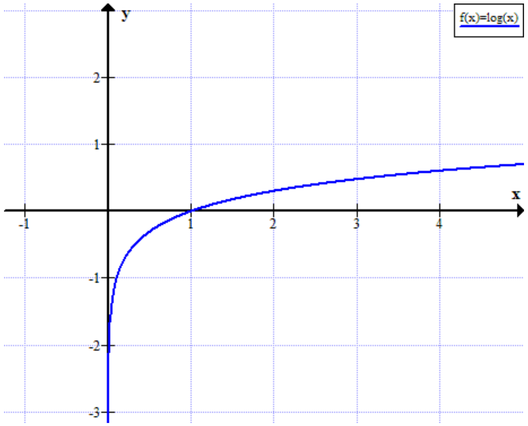
1 hidden layer

$$
\begin{tikzcd}[column sep=small]
x \arrow[dr] & & & W^{[2]} \arrow[d] & & \\
W^{[1]} \arrow[r] & \boxed{z^{[1]} = W^{[1]}x + b^{[1]}} \arrow[r] & \boxed{a^{[1]} = \sigma(z^{[1]})} \arrow[r] & \boxed{z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}} \arrow[r] & \boxed{a^{[2]} = \sigma(z^{[2]})} \arrow[r] & \boxed{\mathcal{L}(a^{[2]}, y)} \\
b^{[1]} \arrow[ur] & & & b^{[2]} \arrow[u] & &
\end{tikzcd}
$$

Logistic regression: función de pérdida

$$L(y, \hat{y}) = -\,(y \log \hat{y} + (1-y) \log (1-\hat{y}))$$

- Si y=1 $\quad L(y,\hat{y}) = -\log \hat{y}$: minimizar $L(y,\hat{y}) \rightarrow$ maximizar $\hat{y}$
- Si y=0 $\quad L(y,\hat{y}) = -\log (1-\hat{y})$: minimizar $L(y,\hat{y}) \rightarrow$ minimizar $\hat{y}$

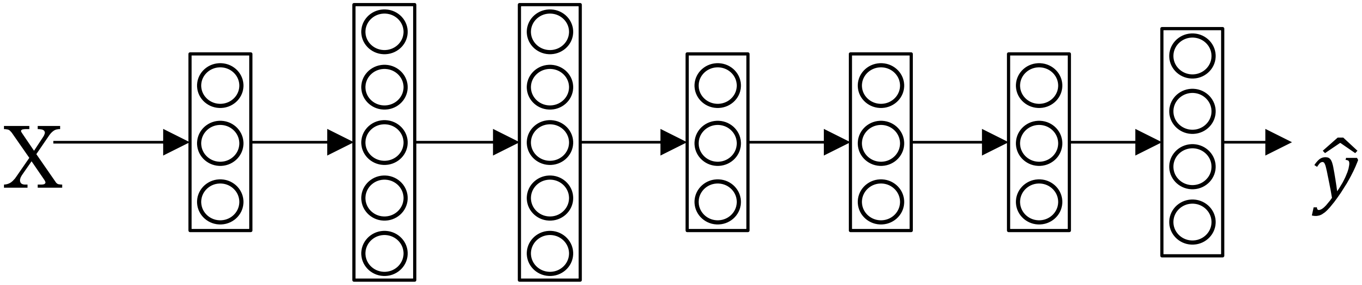


Clasificación multiple: Softmax regression

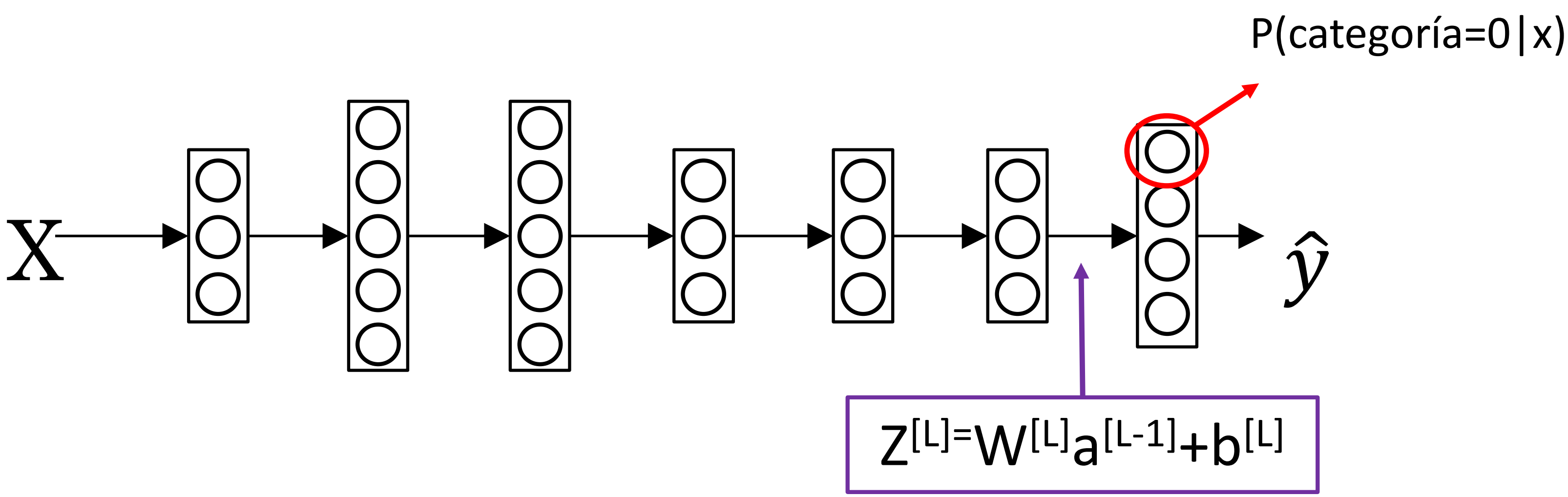
- Generalización de *logistic regression*
- En lugar de clasificar en dos categorías sirve para clasificar en varias

3 1 2 0 3 2 0 1

- La *output layer* tiene el mismo número de unidades que categorías queremos clasificar



Clasificación multiple: Softmax regression



Softmax activation function: $g_i(z^{[L]}) = \dfrac{e^{z_i^{[L]}}}{\sum_j e^{z_j^{[L]}}}$

Peculiaridad:
Toma como input todo el vector $z^{[L]}$

Ejemplo: $z^{[L]} = \begin{matrix} 3 \\ -2 \\ 6 \end{matrix}$

$$g_0(z^{[L]}) = \frac{e^3}{e^3+e^{-2}+e^6} = 0.047$$

$$g_1(z^{[L]}) = \frac{e^{-2}}{e^3+e^{-2}+e^6} = 0.0003$$

$$g_2(z^{[L]}) = \frac{e^6}{e^3+e^{-2}+e^6} = 0.952$$

$$a^{[L]} = \begin{matrix} 0.047 \\ 0.0003 \\ 0.952 \end{matrix} \qquad a^{[L]} = \begin{matrix} 0 \\ 0 \\ 1 \end{matrix}$$

Softmax Hardmax

Clasificación Multiple: Loss function

- En el caso de *logistic regression* usábamos como función de pérdida:

$$L(y,\hat{y}) = -(y \log \hat{y} + (1-y)\log(1-\hat{y}))$$

- Para la clasificación múltiple usamos la versión más general:

$$L(y,\hat{y}) = -\sum_j y_j \log \hat{y}_j$$ ← Cross-Entropy

Ejemplo: $y = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$ = gato $\quad \hat{y} = \begin{pmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{pmatrix} \quad L(y,\hat{y}) = -\log\hat{y}_2$

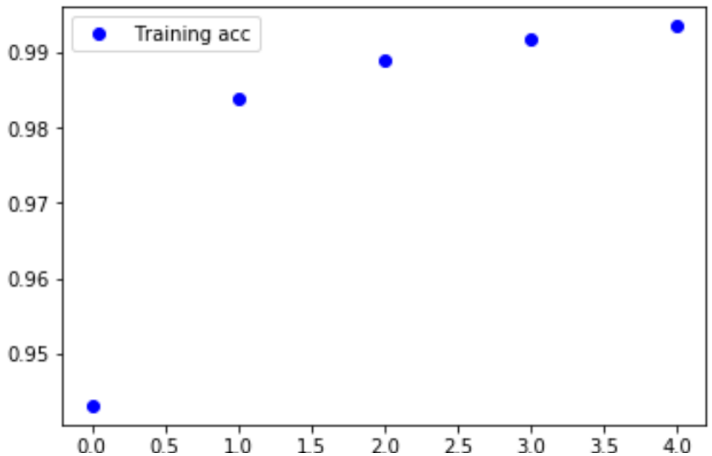
Si queremos que $L(y,\hat{y})$ sea lo más pequeño posible, significa que $\log\hat{y}_2$ (y por tanto $\hat{y}_2$) **tiene que ser lo mayor posible**

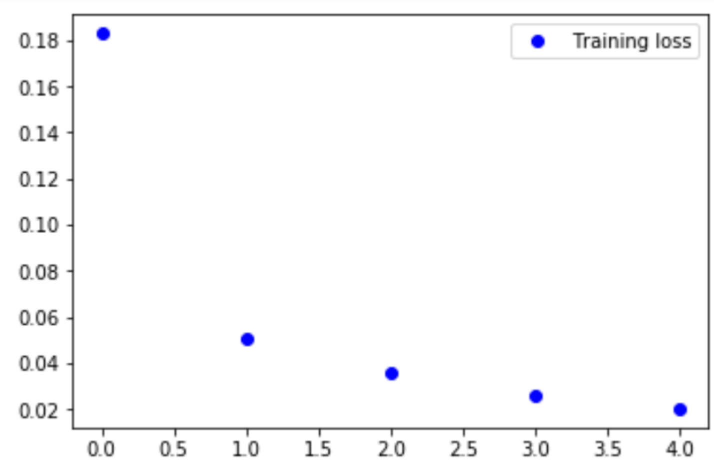
Minimizar $L(y,\hat{y})$ **→ maximizar la probabilidad de la categoría correcta**

$$\boxed{Cost = \frac{1}{m}\sum_{i=1}^{m} L(y_i,\hat{y}_i)}$$

Gato

¿Cómo monitorizar el aprendizaje durante el training?

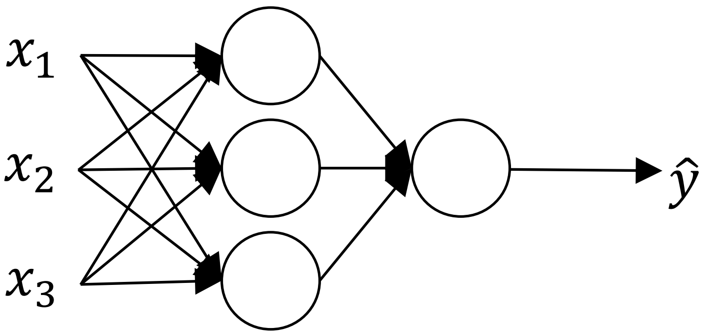




Plots extraídos de la práctica de reconocimiento de números

¿Por qué necesitamos las funciones no lineales (funciones de activación)?

¿Por qué necesitamos las funciones no lineales (funciones de activación)?



Dado un cierto x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[1]} = \cancel{g^{[1]}(z^{[1]})} = z^{[1]}$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = \cancel{g^{[2]}(z^{[2]})} = z^{[2]}$$

$$a^{[1]} = z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[2]} = z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = W^{[2]}(\underbrace{W^{[1]}x + b^{[1]}}_{a^{[1]}}) + b^{[2]}$$

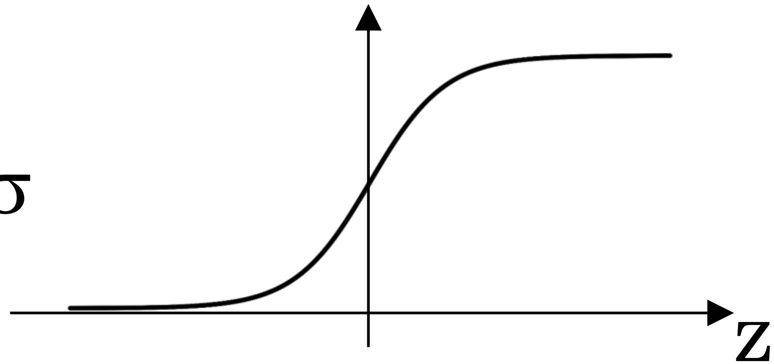
$$= \underbrace{(W^{[2]}W^{[1]})}_{W'}x + \underbrace{(W^{[2]}b^{[1]} + b^{[2]})}_{b'}$$

$$= \underline{W'x + b'}$$

¡Sería equivalente a tener un único nodo!

¿Qué función de activación elegir?

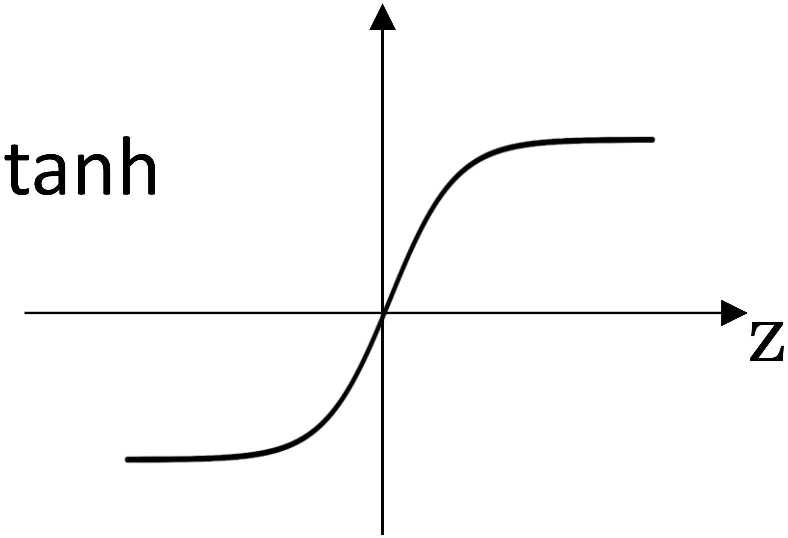
σ



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

$$\sigma'(z) = \sigma(z)\,(1 - \sigma(z))$$

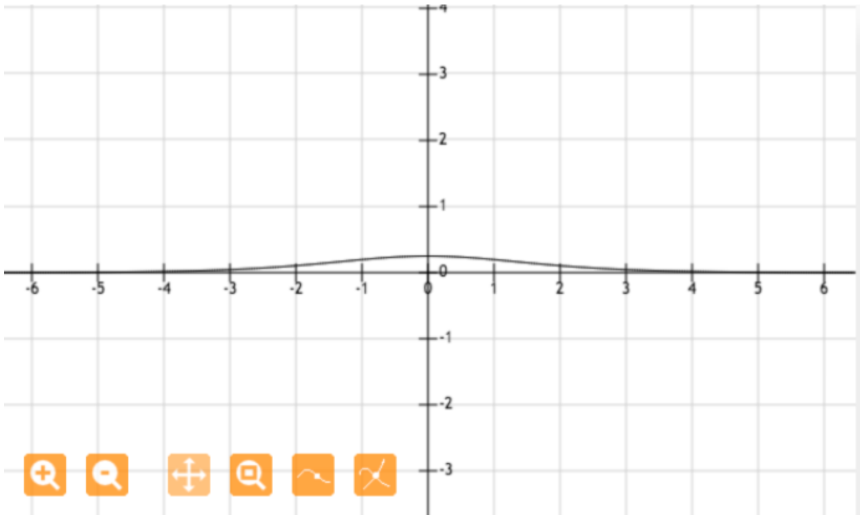
tanh

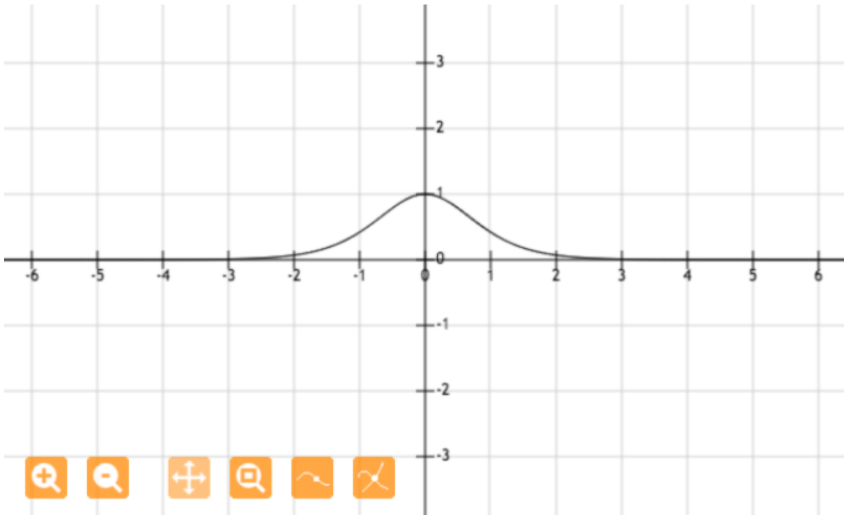


$$g(z) = \tanh(z)$$

$$g'(z) = 1 - \big(\tanh(z)\big)^2$$

¿Qué función de activación elegir?





$$\sigma'(z) = \sigma(z)\,(1 - \sigma(z))$$

$$g'(z) = 1 - \left(\tanh(z)\right)^2$$

¿Qué función de activación elegir?

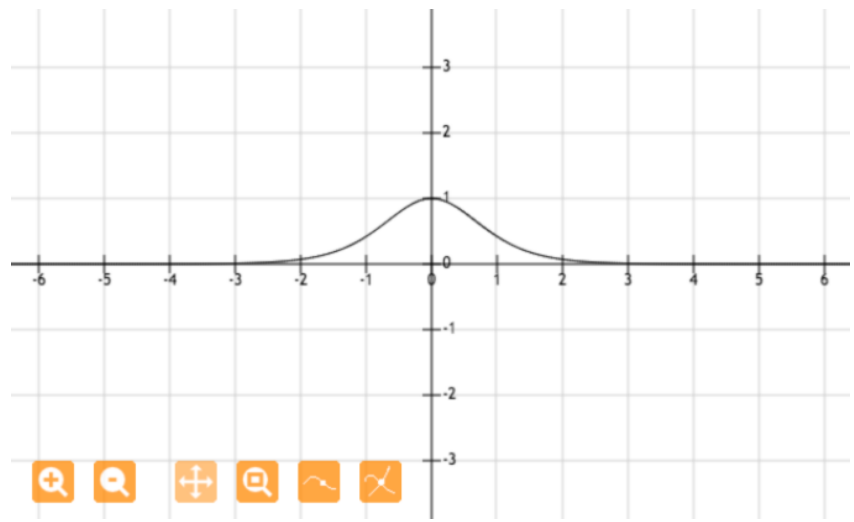
El valor de la derivada es mayor para g(z)= tanh(z) → el aprendizaje es más rápido

En el caso de la sigmoide, el valor de la derivada es como máximo 0.25

Cuando tienes muchas capas, tendrás que multiplicar muchas veces números muy pequeños, dándote un gradiente tan pequeño que prácticamente no aprenderás nada (vanishing gradient).

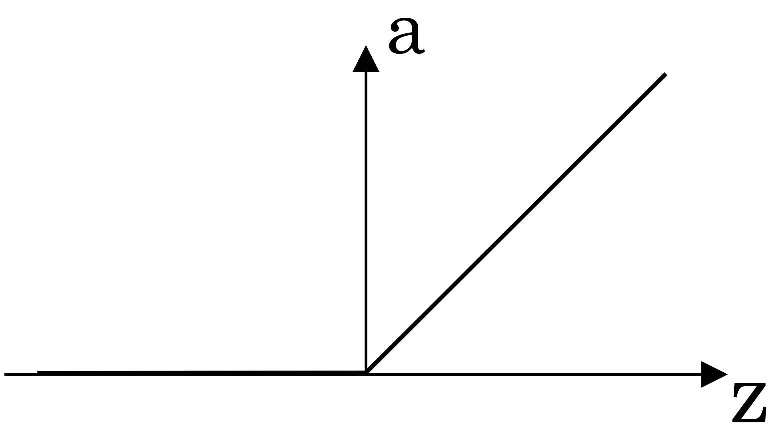
No se puede hacer Deep Learning teniendo sigmoides en las capas ocultas.

tanh(z) funciona mejor (máximo = 1) pero desciende muy rápido.

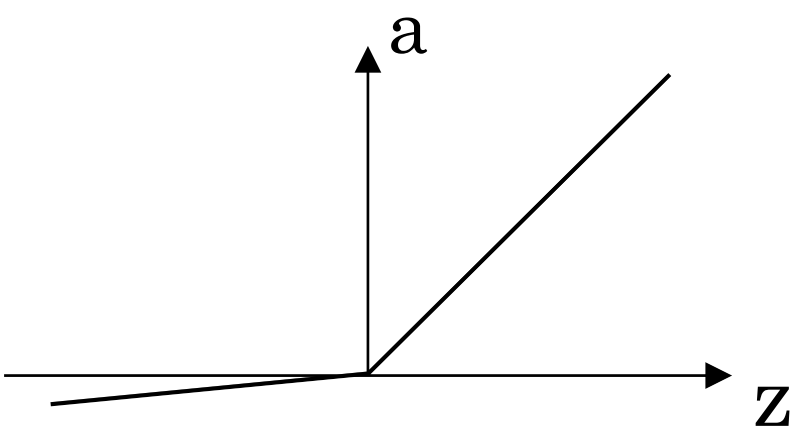


$$g'(z) = 1 - \left(\tanh(z)\right)^2$$

Nuevas funciones de activación



ReLU

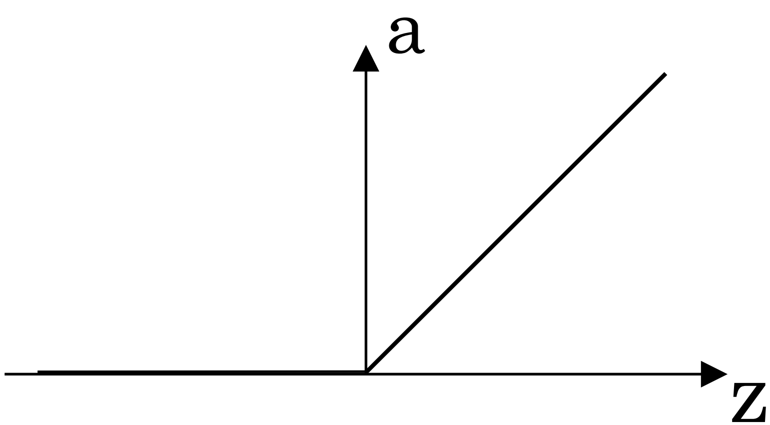


Leaky ReLU

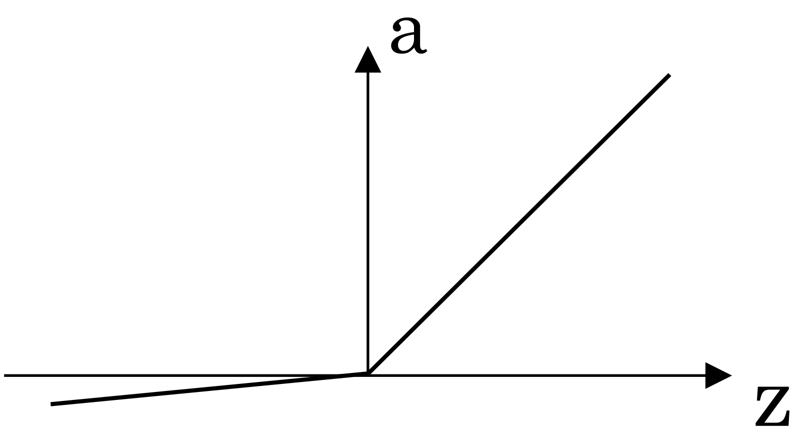
$$f(x) = \max(0, x)$$

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$$

Nuevas funciones de activación



ReLU



Leaky ReLU

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

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$$

Ventajas: Evitamos el problema de *vanishing gradient* ya que la derivada es siempre 1 para x > 0

Desventajas: No es derivable → A efectos prácticos da igual, nunca vamos a tener justo 0

Recetario para las funciones de activación

Cada problema es un mundo, pero podemos establecer un recetario general para las funciones de activación y hacer pruebas usando esta configuración como base:

- Usa ReLU (o leaky ReLU) para todas las capas escondidas
- Usa tangh o sigmoide solo en la última capa cuando estés en un problema de clasificación binaria
- Usa softmax en la última capa cuando estés en un problema de clasificación multiple

Resumiendo

Hasta aquí hemos visto:

- Como funciona la red neuronal más básica: logistic regression (arquitectura y *loss function*)
- Como funciona una red neuronal para clasificación múltiple (arquitectura y *loss function*)
- Para que necesitamos funciones no lineales en las redes neuronales y cuales son las más populares

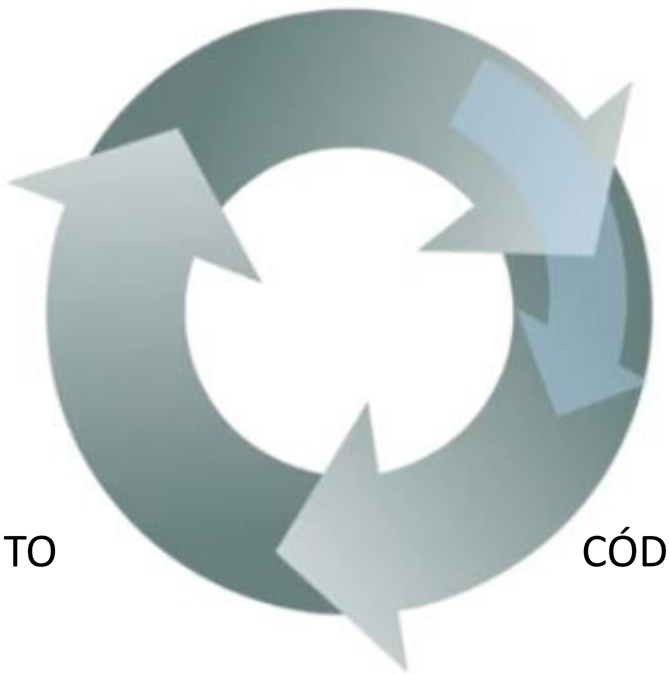
Deep Learning práctico: Dataset, problemas y optimización

Deep Learning: proceso iterativo

Hiperparámetros

- Número de capas
- Número de unidades en cada capas
- Learning rate
- Funciones de activación

…



Datasets

Datos	Training set	Dev set	Test set

- Entrenar usando el **Training Set** usando diferentes modelos
- Probarlos en **Development Set** y elegir el modelo que de mejores resultados
- Una vez elegido este modelo probarlo en el **Test Set** para tener una estimación sin sesgo de como de bien funciona

Antes de Big Data: 70% (train) / 30% (test) ~10.000
60% (train) / 20% (dev) / 20% (test)

Big Data: 98% (train) / 1% (dev) / 1% (test) ~1.000.000
10000!

Datasets

Problema típico : Diferentes tipos de datasets

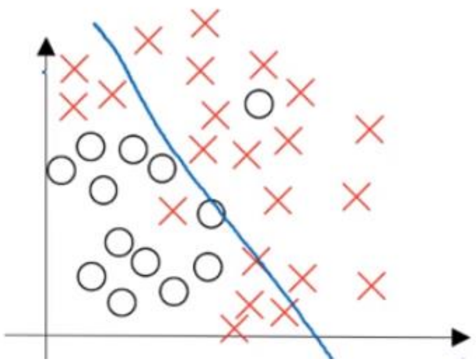
Entrenas tu red con
fotos de la web (alta resolución)

Dev/Test datasets
con imágenes de móvil

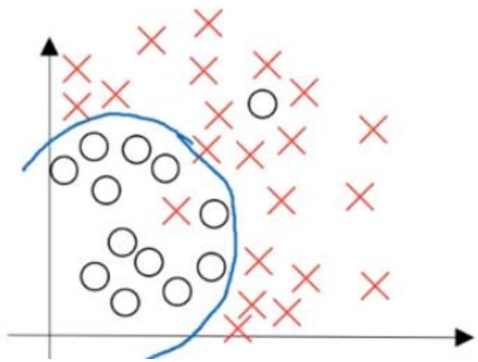
No hay ningún problema mientras Dev y Test tengan el mismo tipo de imágenes

Varianza/Sesgo

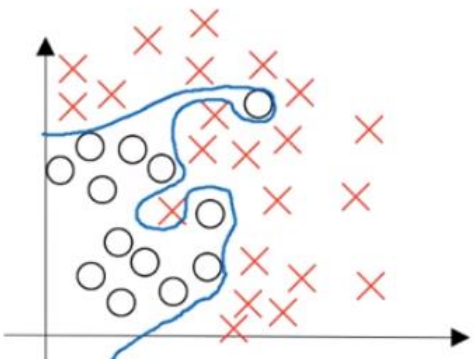
REPASO



Sesgo alto
(underfitting)



"Bien"



Varianza alta
(overfitting)

Por desgracia, no es fácil representar problemas de altas dimensionalidad (como las imágenes) para poder ver cual es la línea de separación entre categorías.

Pero existen otras métricas

Varianza/Sesgo

REPASO

y=0

y=1

Error humano ~ 0%

Error óptimo (Error de Bayes)

Train set error	1%	15%	15%	0.5%
Dev set error	11%	16%	30%	1%

Varianza/Sesgo

REPASO

y=0

y=1

Error humano ~ 0%

Train set error	1%	15%	15%	0.5%
Dev set error	11%	16%	30%	1%
	Varianza Alta	Sesgo alto	Sesgo alto Varianza Alta	Sesgo bajo Varianza baja

Error óptimo (Error de Bayes)

Varianza/Sesgo

REPASO

y=0

y=1

Error humano ~ 0%

Train set error	1%	15%	15%	0.5%
Dev set error	11%	16%	30%	1%
	Varianza Alta	Sesgo alto	Sesgo alto Varianza Alta	Sesgo bajo Varianza baja

Error óptimo (Error de Bayes)

Y si el error óptimo hubiera sido 15% ?

Varianza/Sesgo

REPASO

y=0

y=1

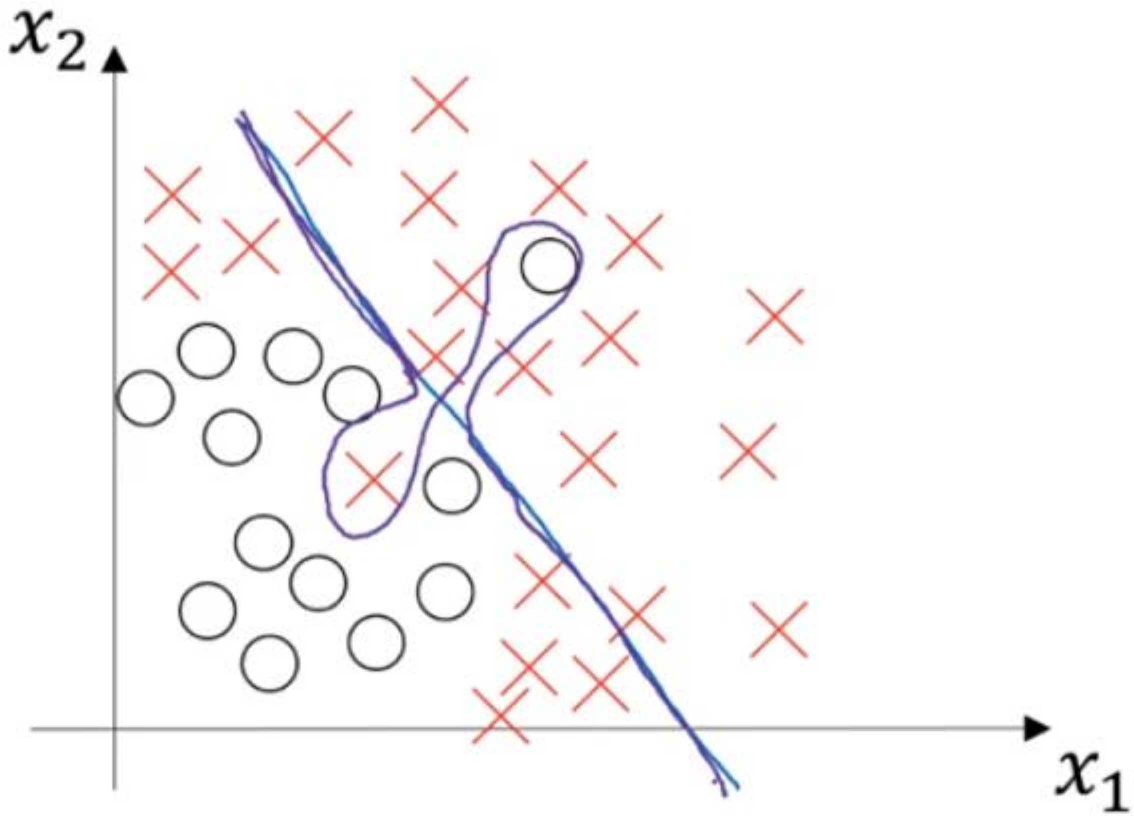
Error humano ~ 0%

Train set error	1%	15%	15%	0.5%
Dev set error	11%	16%	30%	1%
		Sesgo bajo Varianza baja		

Error óptimo (Error de Bayes)

Y si el error óptimo hubiera sido 15% ?

Varianza alta y Sesgo alto



Receta

¿Sesgo alto?
(training data performance)

Sí

- Red más grande
- Entrenar más tiempo
- Búsqueda de otra arquitectura NN

No

¿Varianza alta?
(dev set performance)

Sí

- Más datos
- Regularización (coming soon!)
- Búsqueda de otra arquitectura NN

No

Regularización: L2

- Si tienes un problema varianza alta (*overfitting*) una de las primeras cosas para probar es conseguir más datos → No siempre es posible.
- En ese caso, el siguiente paso es aplicar algún tipo de **regularización.**

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L\left(\hat{y}^{(i)}, y^{(i)}\right) + \frac{\lambda}{2m}\|w\|^2 \qquad w \in R^{n_x} \qquad b \in R$$

- Regularización L2: $\|w\|^2 = \sum w_j^2 = w^T w$
- **λ:** parámetro de regularización → Se elige en el dev set

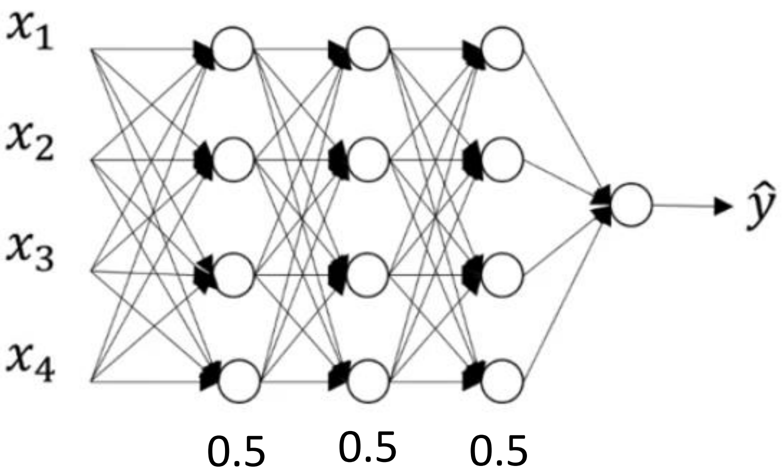
Regularización: L2

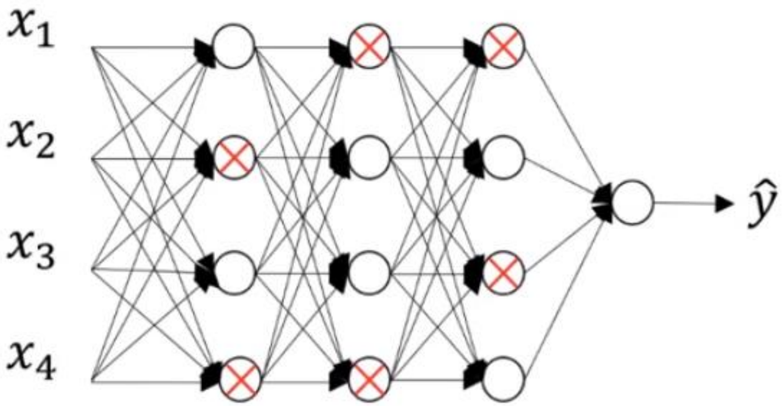
$$J(w,b) = \underbrace{\frac{1}{m}\sum_{i=1}^{m} L\left(\hat{y}^{(i)}, y^{(i)}\right)}_{\text{Ajustarse al modelo lo mejor posible}} + \underbrace{\frac{\lambda}{2m}\|w\|^2}_{\text{Hacer que los w sean lo menor posibles}}$$

Si los pesos w son lo más pequeños posibles, la red tenderá a ser menos compleja

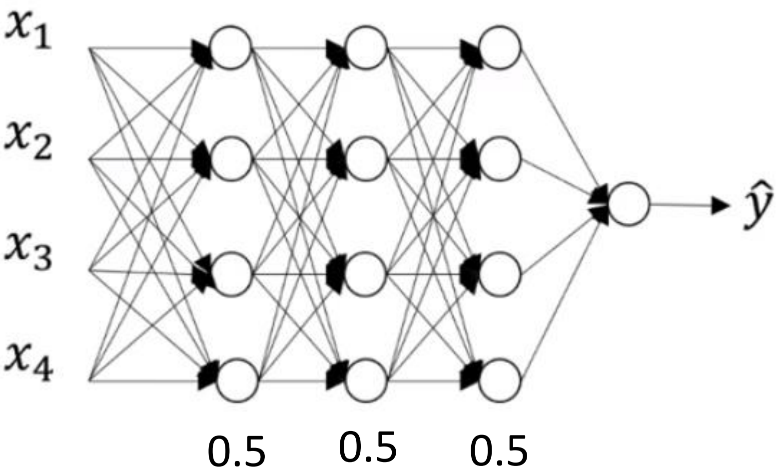
Menos especializada

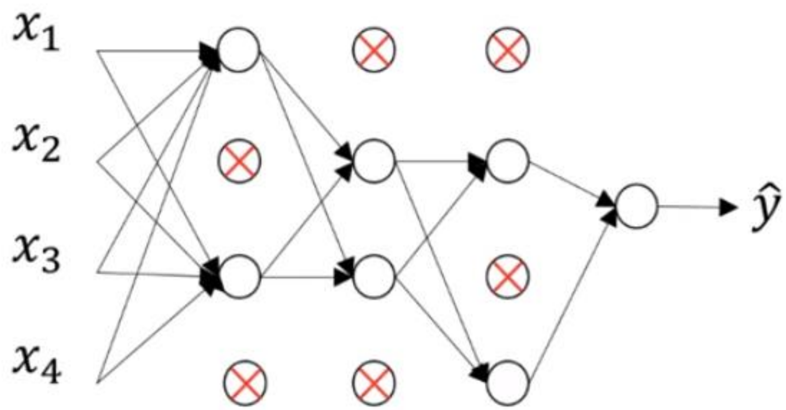
Regularización: Dropout





Regularización: Dropout

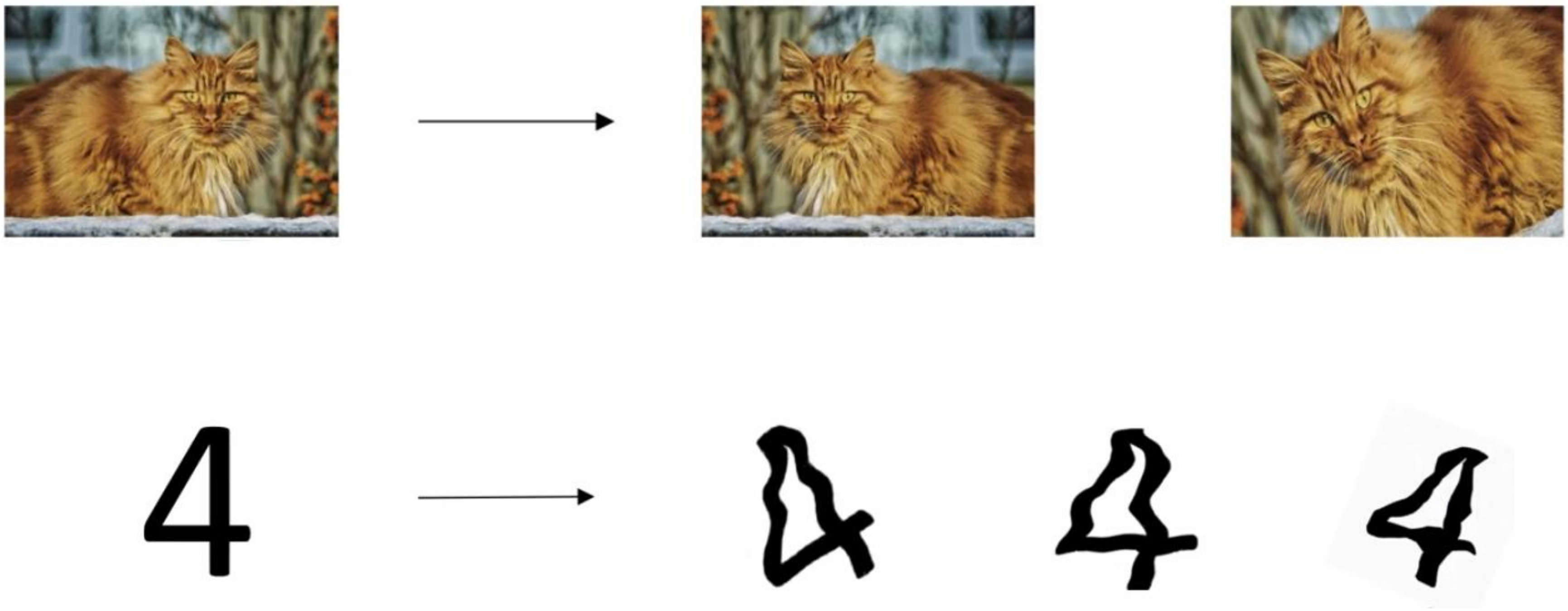




- Simplificas la red
- Evitas que algún nodo se "sobre-especialice"

No puede depender de un input concreto porque éste puede desaparecer en cualquier momento

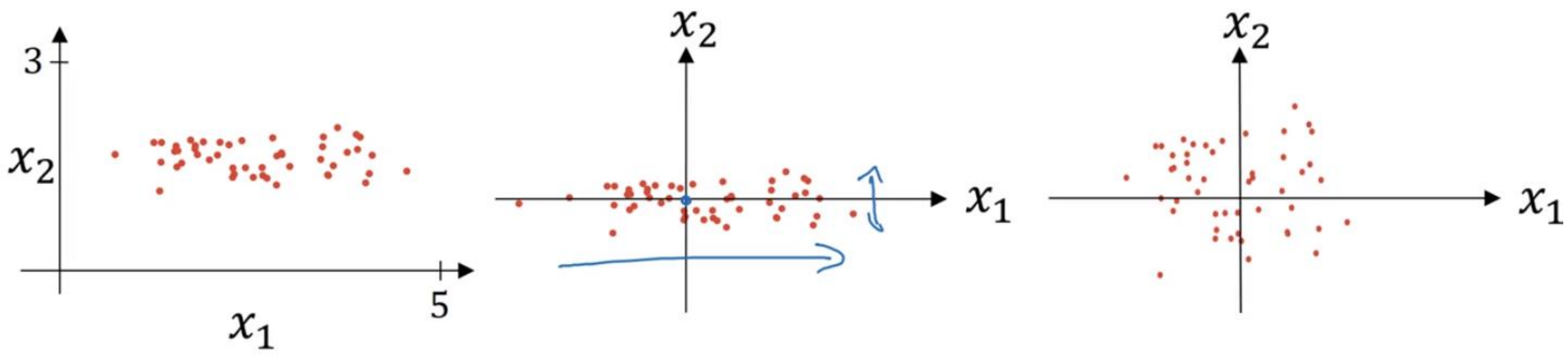
Regularización: Data Augmentation



No funciona tan bien como conseguir más datos independientes pero es gratis ☺

Normalización del input

- Input $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$



Substraemos la media:

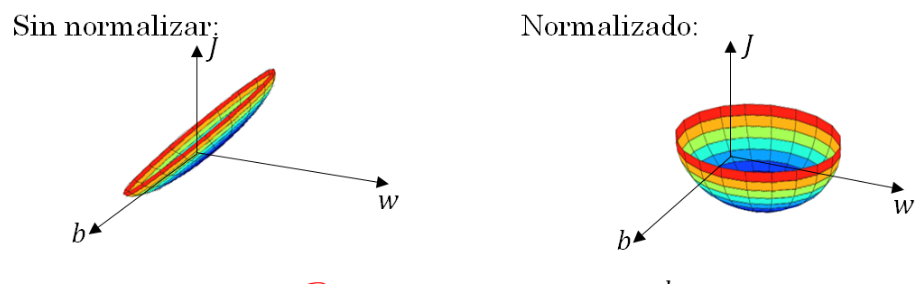
$$\mu = \frac{1}{m} \sum_{I=1}^{m} x^{(i)} \qquad x = x - \mu$$

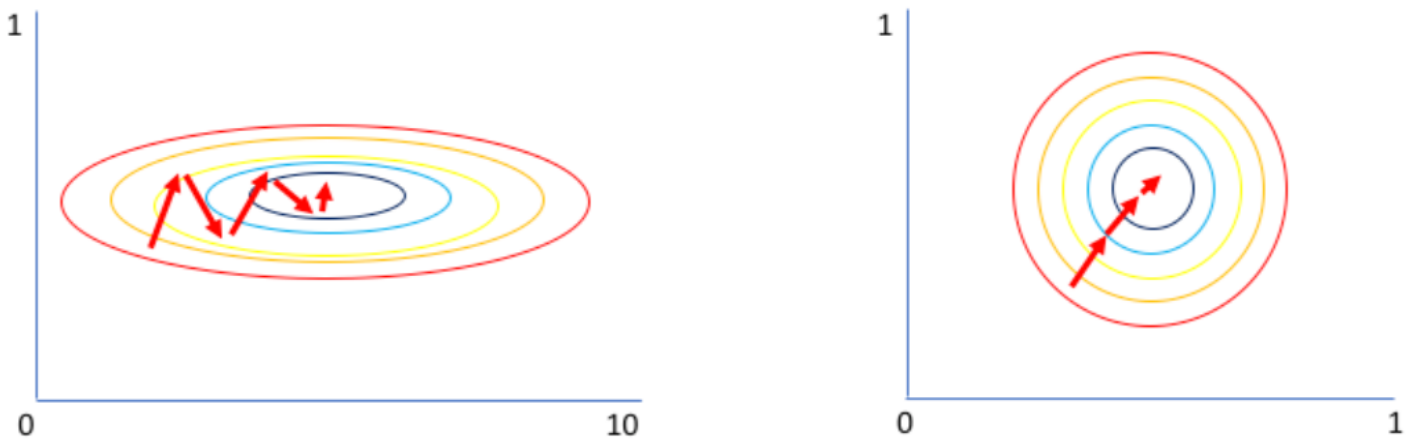
Normalizar por la desviación estándar:

$$\sigma^2 = \frac{1}{m} \sum_{I=1}^{m} x^{(i)\,2} \qquad x\ /= sqrt(\sigma^2)$$

Normalización del input

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right)$$





Gradient of larger parameter dominates the update

Both parameters can be updated in equal proportions

Inicialización de los pesos

$$z = w_1x_1 + w_2x_2 + w_3x_3 + \ldots + w_nx_n$$

var(Z) = var(X) → Si estás usando ReLU, para conseguir esto:

$W^{[l]}$ = np.random.randn(shape)* np.sqrt ($2/n^{[l-1]}$)

Inicialización de pesos Xavier (sigmoide y tanh)

"Understanding the difficulty of training deep feedforward neural networks"

Inicialización de pesos He (ReLU)

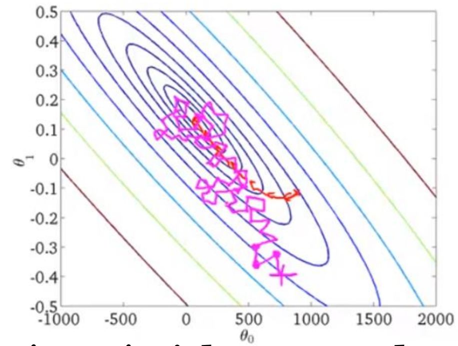
"Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification"

Stochastic Gradient Descent

- Batch gradient descent: se actualizan los pesos tras haber recorrido todo el dataset (una época entera)

Stochastic gradient descent:

- Los pesos se actualizan para cada muestra del dataset de training
- Cada actualización intenta mejorar la predicción para ajustarse a una única muestra
- Cada iteración será extremadamente rápida
- Conviene hacer un *random shuffle* de los datos



- El stochastic gradient descent puede ser mucho más rápido que el batch gradient descent para datasets muy grandes
- Puede tener problemas convergiendo, una posible solución:

$$\alpha = \frac{const\_1}{n_{iteración} + const\_2}$$

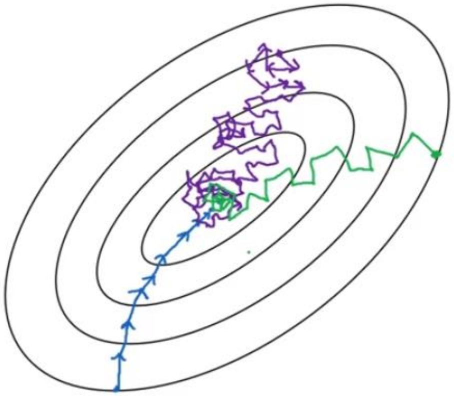
Mini-Batch gradient descent

- Batch gradient descent: se actualizan los pesos tras haber recorrido todo el dataset (una época entera)
- Stochastic gradient descent: se actualizan los pesos muestra a muestra

Mini-batch gradient descent:

- Dividir el dataset de entrenamiento en fragmentos más pequeños (mini-batches)
- Los parámetros de la red se actualizan para cada mini-batch.
- Si tienes una buena vectorización el mini-batch gradient descent puede ser más rápido que el stochastic (paralelización)
- El tamaño del mini-batch depende de tu muestra

Stochastic, Mini-batch y Batch



Stochastic gradient descent

Actualizas los parámetros sin tener que recorrer toda la muestra

Se pierde la potencia de la vectorización

Puede tener problemas de convergencia en una región relativamente grande

Mini-Batch gradient descent:

Aprendizaje más rápido:

Actualizas los parámetros sin tener que recorrer toda la muestra

Explotas la vectorización

Mejor convergencia

Batch gradient descent:

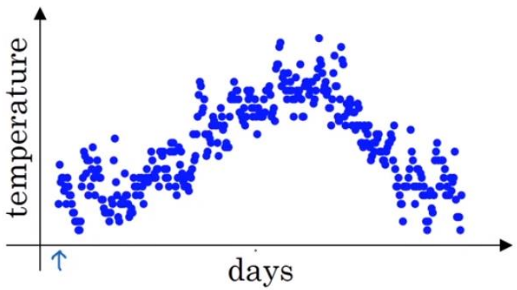
Si la muestra de entrenamiento es muy grande, cada iteración lleva mucho tiempo

¿Cómo elegir el tamaño del mini-batch?

- Si tienes una muestra pequeña: usa Batch Gradient Descent (m < 2000)

- Si tienes una muestra grande:
  - Típicos tamaños para el mini-batch: 64, 128, 256…1024
  - Por como funciona el acceso a memoria en los ordenadores suele funcionar mejor que sea una potencia de 2
  - Estar seguro de que tu mini-batch entra dentro de la memoria de la CPU/GPU
  - En la práctica el tamaño del mini-batch es otro hiperparámetro: hacer pruebas con unos cuantos y decidir cuál es mejor

Media Móvil Exponencial



- Si quieres calcular la tendencia (media móvil) de la temperatura:

$$V_t=\beta V_{t-1} + (1-\beta)\theta_t$$

V_t es como sacar la media de temperatura de los últimos $\frac{1}{1-\beta}$ días

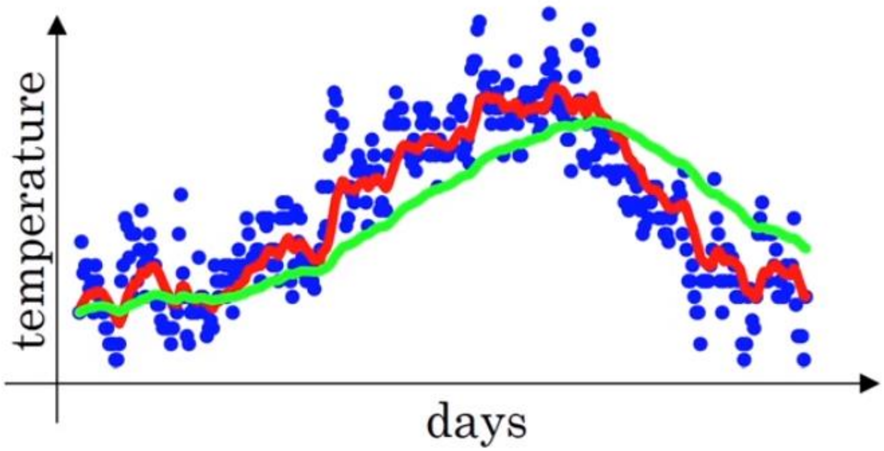
$\beta = 0.9 \sim 10$ días
$\beta = 0.98 \sim 50$ días
$\beta = 0.5 \sim 2$ días

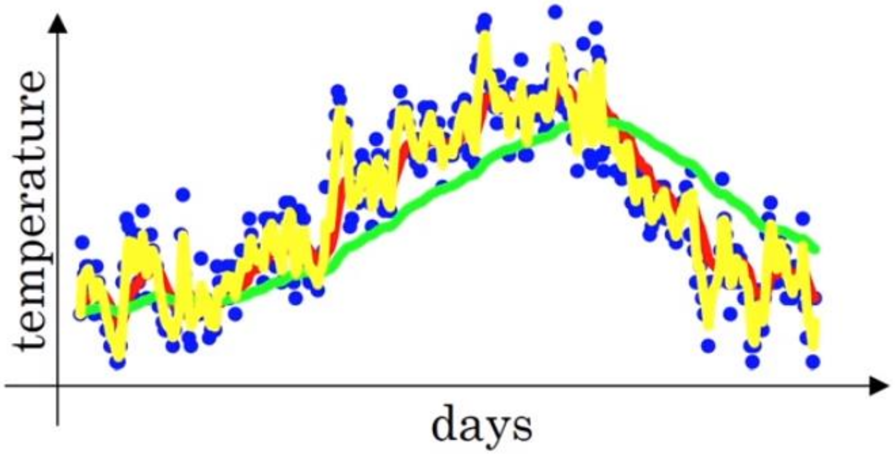
Si tienes un valor de β muy alto → se adapta mal a cambios bruscos ya que da más peso a la temperatura anterior que a la presente (latencia)

Si tienes un valor de β muy bajo → Más susceptible de dejarse influir por valores atípicos

Media Móvil Exponencial

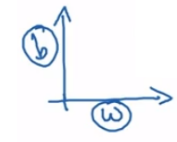
$\beta = 0.9 \sim 10$ días
$\beta = 0.98 \sim 50$ días
$\beta = 0.5 \sim 2$ días

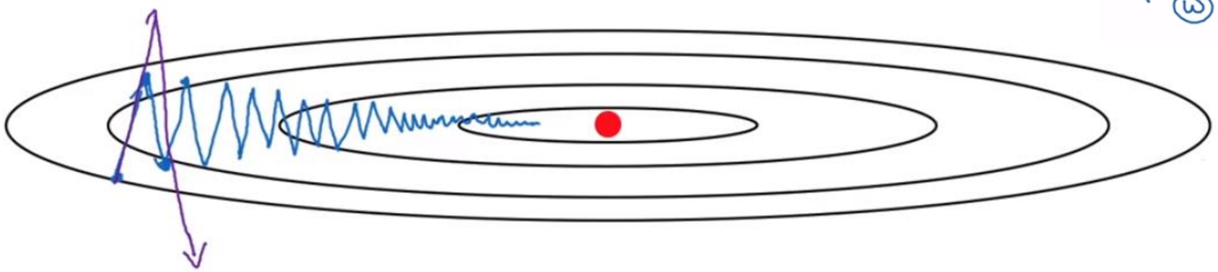




Gradient Descent con momento

- Usar la media móvil exponencial de los gradientes y actualizar los pesos con ese valor
- Casi siempre funciona más rápido que el *mini-batch gradient descent* normal





Las oscilaciones hacen que no puedas coger un α grande

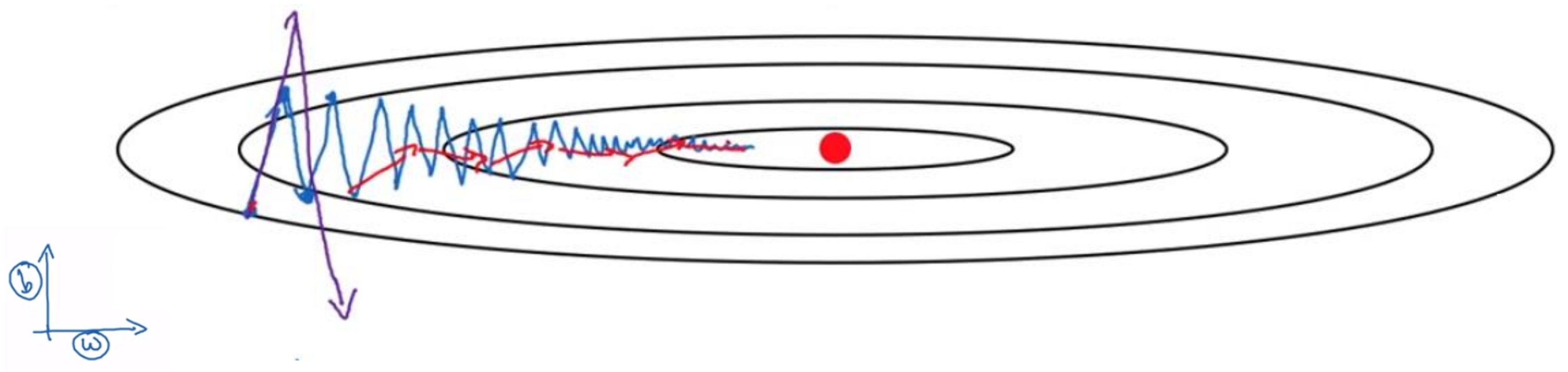
Aprendizaje más rápido

Aprendizaje más lento

En la iteración sobre cada mini-batch:

$$Calcula\ dW\ y\ db$$
$$v_{dW} = \beta v_{dW} + (1-\beta)dW$$
$$v_{db} = \beta v_{db} + (1-\beta)db$$
$$W = W - \alpha v_{dW} \quad b = b - \alpha v_{db}$$

Gradient Descent con momento



- Hiperparámetros: α y β
- El valor más comúnmente usado es β=0.9 (~ últimos 10 gradientes)
- Se puede probar para ve si hay algún otro valor que funcione mejor

Otros Algoritmos de Optimización

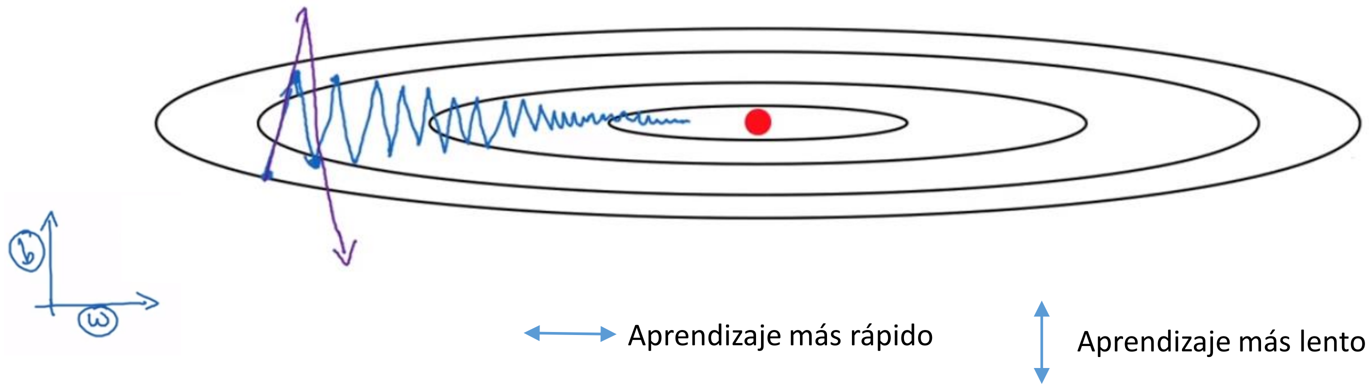
- Muchos de los algoritmos de optimización no generalizan bien → reticencia frente a los nuevos algoritmos

- De los pocos que han conseguido hacerse un camino frente al *gradient descent con momento*.

RMSProp

Adam Optimization

RMSProp



En la iteración sobre cada mini-batch:

$$Calcula\ dW\ y\ db$$

$$s_{dW} = \beta s_{dW} + (1-\beta)dW^2 \leftarrow \boxed{\text{pequeño}}$$

$$s_{db} = \beta s_{db} + (1-\beta)db^2 \leftarrow \boxed{\text{grande}}$$

$$W = W - \alpha \frac{dW}{\sqrt{s_{dW}} + \varepsilon} \quad b = b - \alpha \frac{db}{\sqrt{s_{db}} + \varepsilon}$$

Consigues regular la oscilación → puedes usar α mayores

Adam

Ponemos juntos

Gradient descent con momento

Gradient descent con RMSProp

Adam* Optimization

*Adaptive Moment estimation

Adam Optimization

$v_{dw}=0$, $s_{dw}=0$, $v_{db}=0$, $s_{db}=0$

En la iteración sobre cada mini-batch:

Calculamos dW, db

$$\left.\begin{aligned} v_{dW} &= \beta_1 v_{dW} + (1-\beta_1)dW \\ v_{db} &= \beta_1 v_{db} + (1-\beta_1)db \end{aligned}\right\} \text{Momento}$$

$$\left.\begin{aligned} s_{dW} &= \beta_2 s_{dW} + (1-\beta_2)dW^2 \\ s_{db} &= \beta_2 s_{db} + (1-\beta_2)db^2 \end{aligned}\right\} \text{RMSProp}$$

$$W = W - \alpha \frac{v_{dW}}{\sqrt{s_{dW}} + \varepsilon} \qquad b = b - \alpha \frac{v_{db}}{\sqrt{s_{db}} + \varepsilon}$$

Adam Optimization

Hiperparámetros:

α : hay que tunearlo

β_1: 0.9

β_2: 0.999

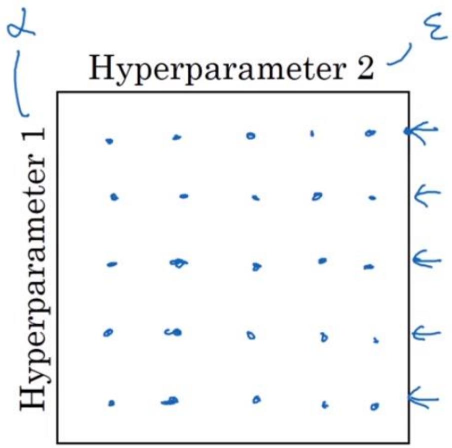
ε: 10^{-8}

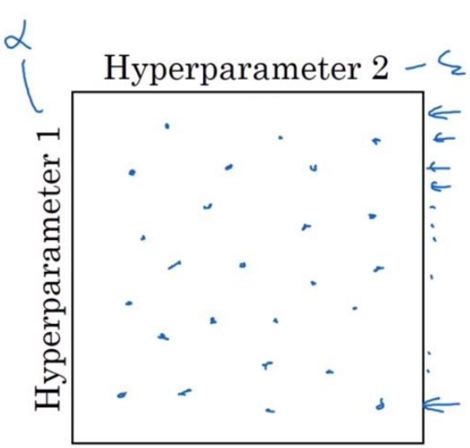
Recomendaciones del artículo original de Adam

Tuneando los hiperparámetros

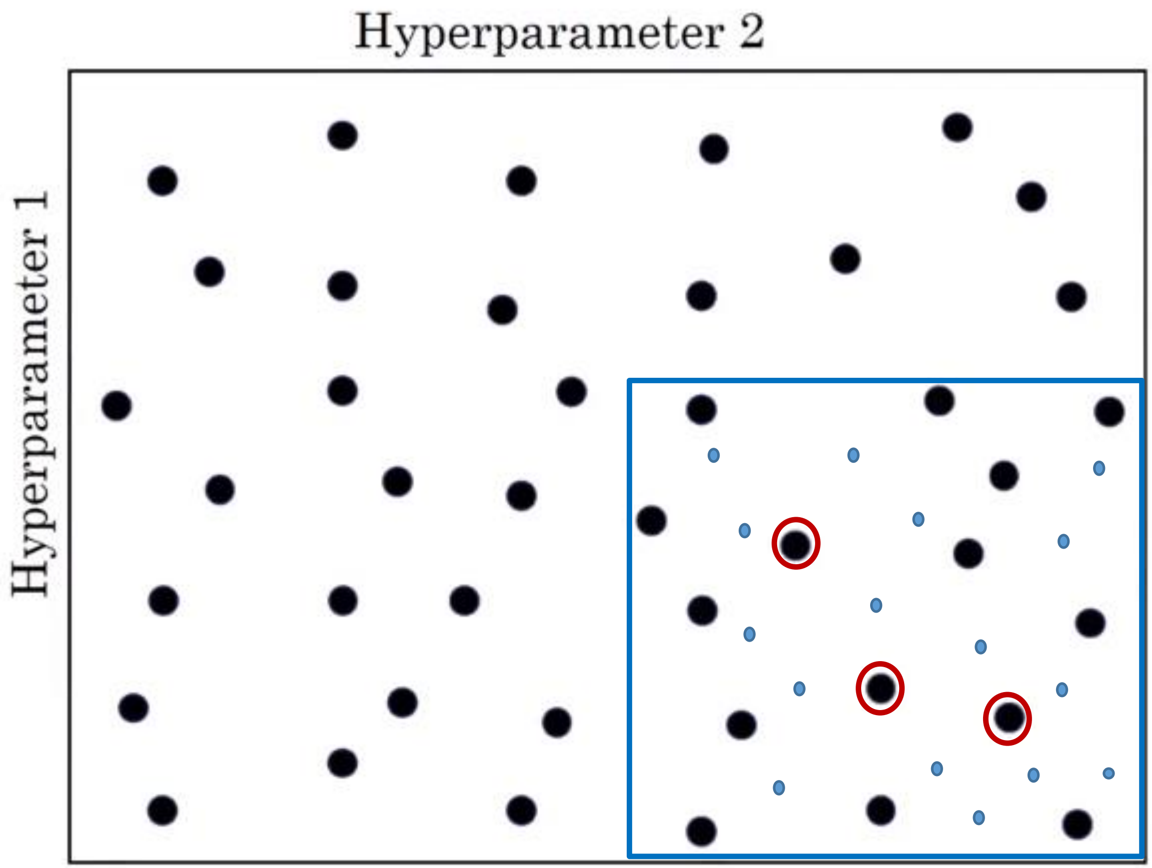
- Learning rate α
- β
- β_1, β_2, ε…
- Número de capas
- Número de nodos en cada capa oculta
- Tamaño del minibatch

…etc





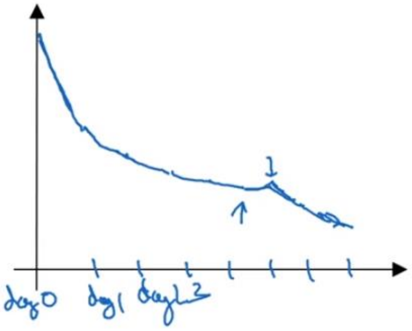
Tuneando los hiperparámetros

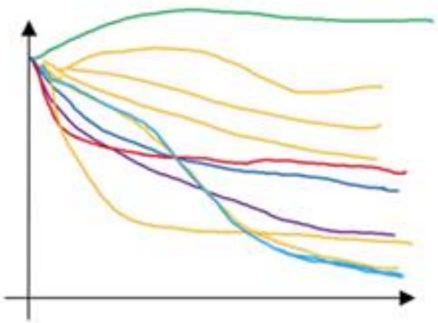


Panda vs Caviar

No tienes muchos recursos → Hacer de "niñera" para un único modelo

Muchos recursos → Entrenas varios modelos y ves cual va mejor



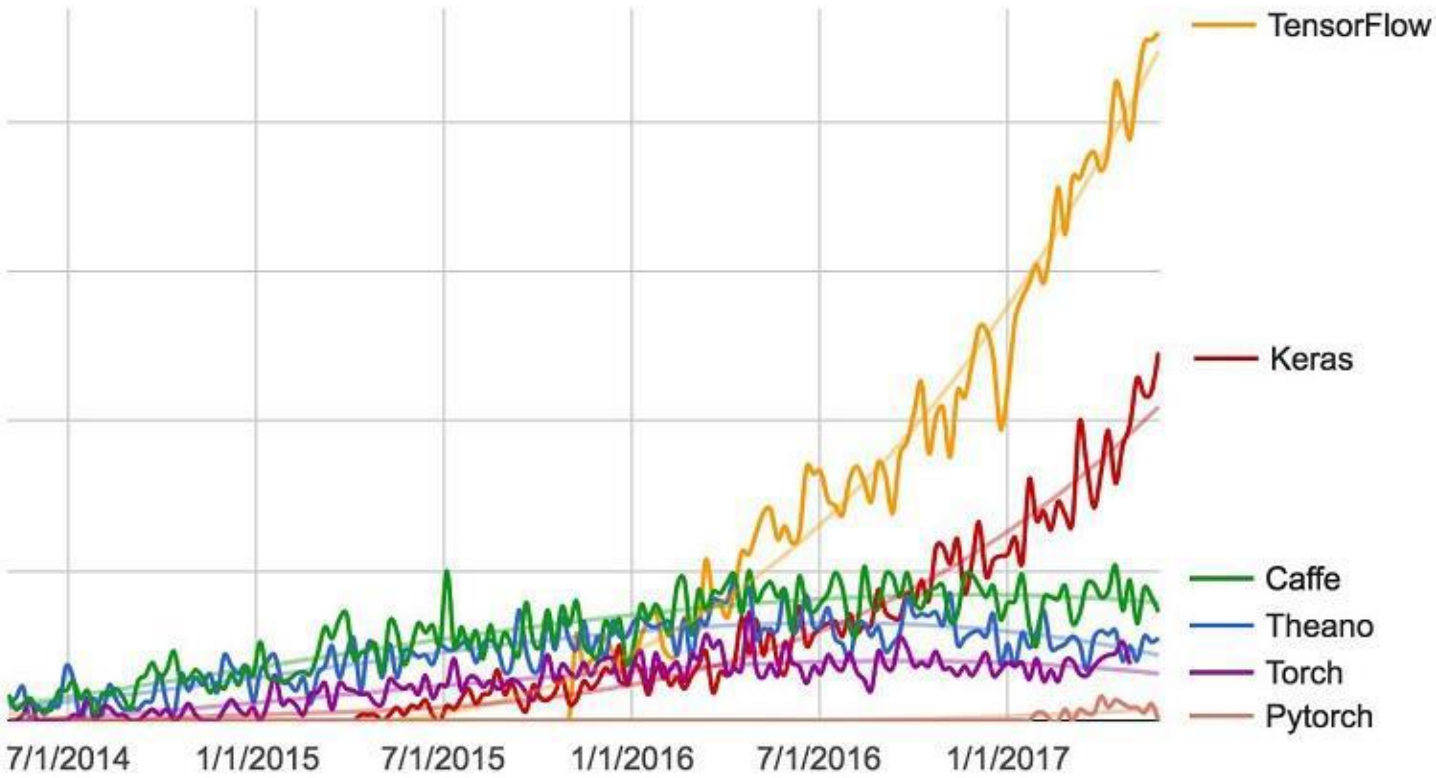


Frameworks para Deep Learning

- Caffe/Caffe2
- CNTK
- DL4J
- Keras
- Lasagne
- mxnet
- PaddlePaddle
- TensorFlow
- Theano
- Torch

- Facilidad para programar
- Velocidad
- Que sea verdaderamente Open Source ☺

Frameworks para Deep Learning



Resumiendo

- Como dividir el dataset de entrenamiento
- Repaso de varianza y sesgo
- Regularización: L2, dropout, data augmentation
- Minibatch y stochastic gradient descent
- Algoritmos de optimización: Momento, RMSProp y Adam
- Cómo *tunear* hiperparámetros

Y mientras tanto en la prensa…

Facebook ha apagado una inteligencia artificial que había «cobrado vida»

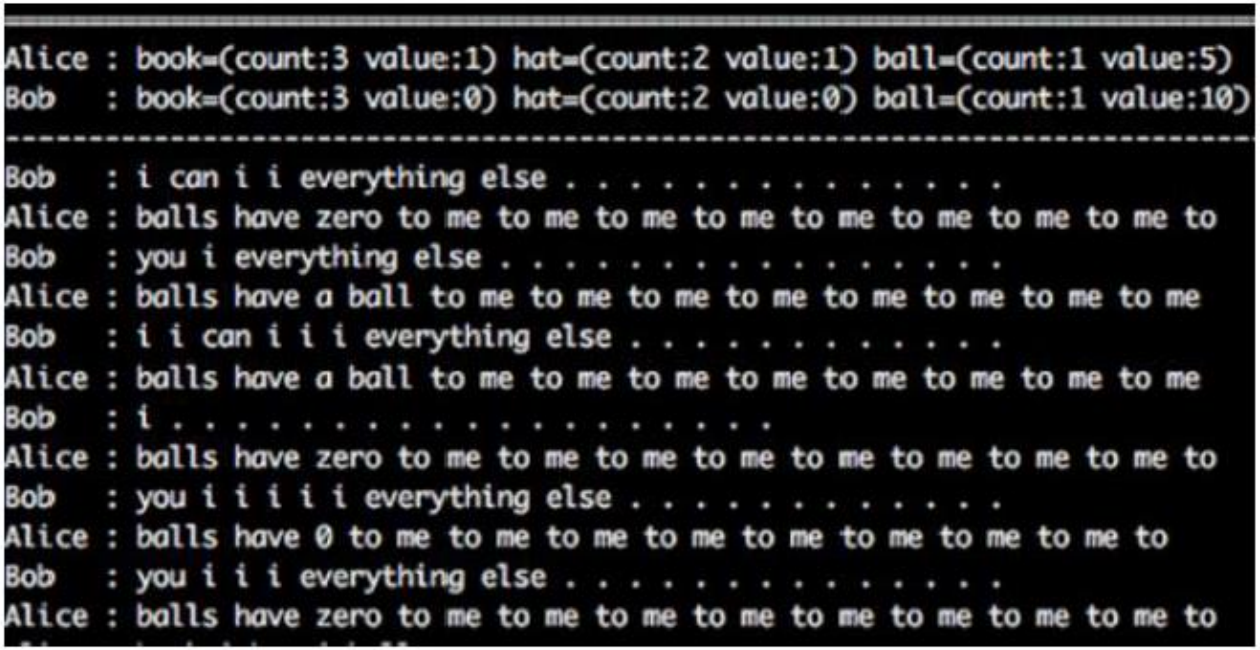
- Investigadores de la división Facebook Artificial Intelligence Researchers desactivan dos «bots» que habían creado su propio lenguaje por su cuenta

- Los bots estaban diseñados para aprender a negociar
- El equipo de facebook no introdujo ninguna recompensa por utilizar el inglés como lengua de comunicación….
- Así que los bots encontraron una representación más óptima para su propósito.

Y mientras tanto en la prensa…

Facebook ha apagado una inteligencia artificial que había «cobrado vida»

- Investigadores de la división Facebook Artificial Intelligence Researchers desactivan dos «bots» que habían creado su propio lengua



- Los bots
- El equipo … no lengua de comu
- Así que lo

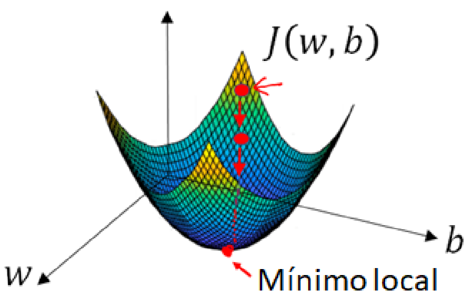
Backup

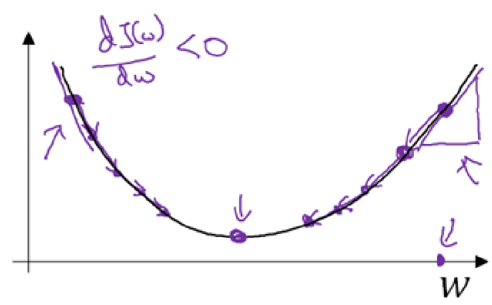
Repaso: Minimización

$$\hat{y} = \sigma(w^T x + b), \quad \sigma(z) = \frac{1}{1+e^{-z}}$$

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

Want to find w, b that minimize $J(w,b)$





$$w := w - \alpha \frac{dJ(w,b)}{dw}$$

$$b := b - \alpha \frac{dJ(w,b)}{db}$$