

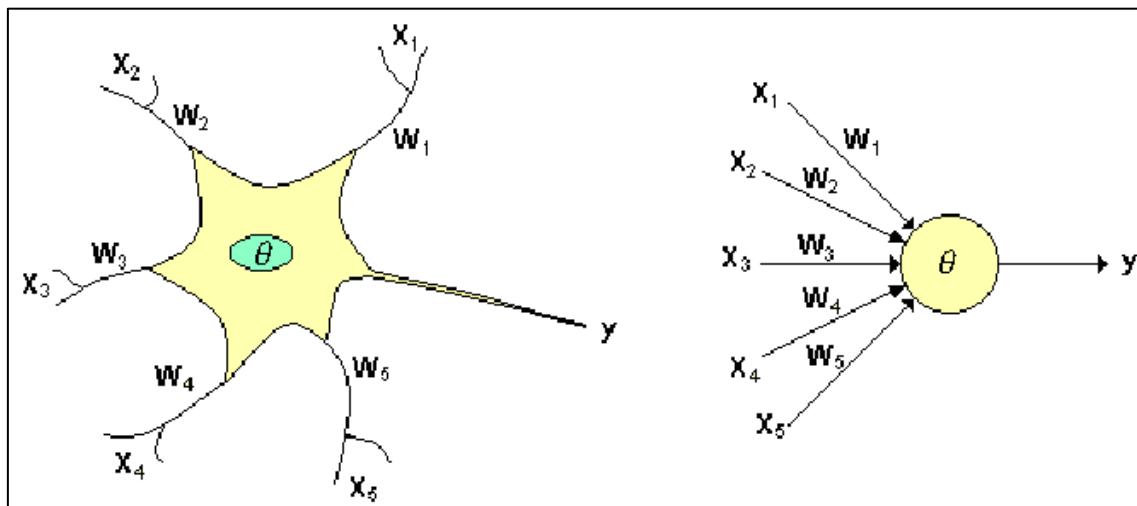
# Resumen materia redes neuronales

## Redes neuronales artificiales: repaso

Su funcionamiento se encuentra inspirado en el cerebro humano en lo que se refiere a:

- Procesamiento de la información proveniente del entorno en tiempo real.
- Robustez y tolerancia a fallas
- Capacidad de adaptación.
- Manejo de información difusa, con ruido e inconsistente.
- Procesamiento paralelo.

## Similitudes entre una neurona biológica y una neuronal artificial



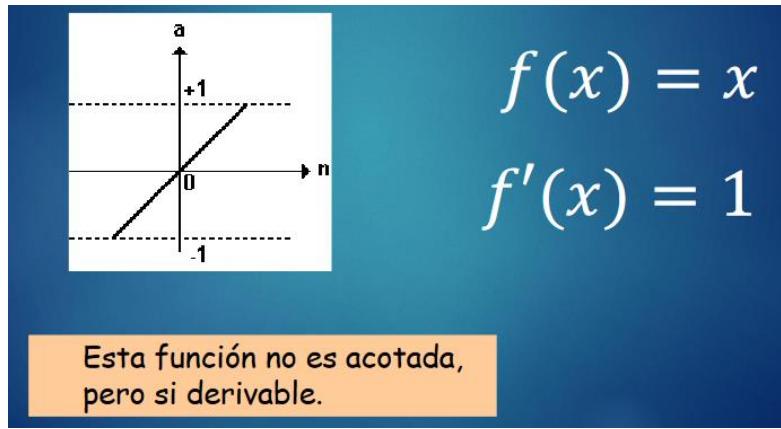
- Las entradas  $X_i$  representan las señales que provienen de otras neuronas y que son capturadas por las dendritas
- Los pesos  $W_i$  son la intensidad de la sinápsis que conecta dos neuronas; tanto  $X_i$  como  $W_i$  son valores reales.
- Theta es la función umbral que la neurona debe superar para activarse; este proceso ocurre biológicamente en el cuerpo de la célula

La entrada neta de una neurona puede escribirse de la siguiente manera:

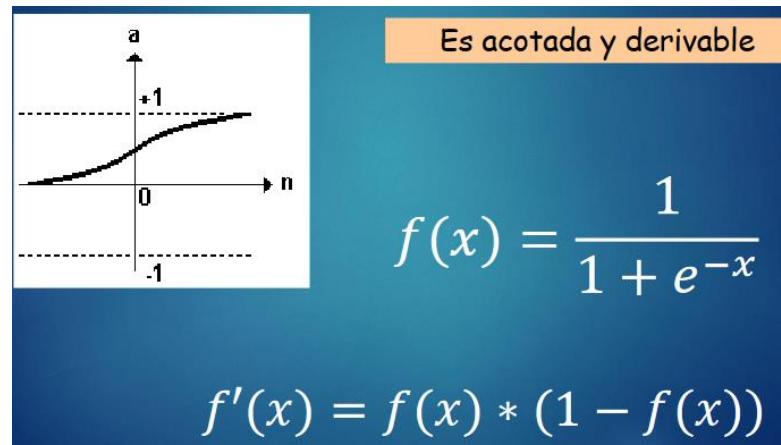
$$neta_j = \sum_{i=1}^n x_i w_i = X \cdot W$$

## Funciones de transferencia

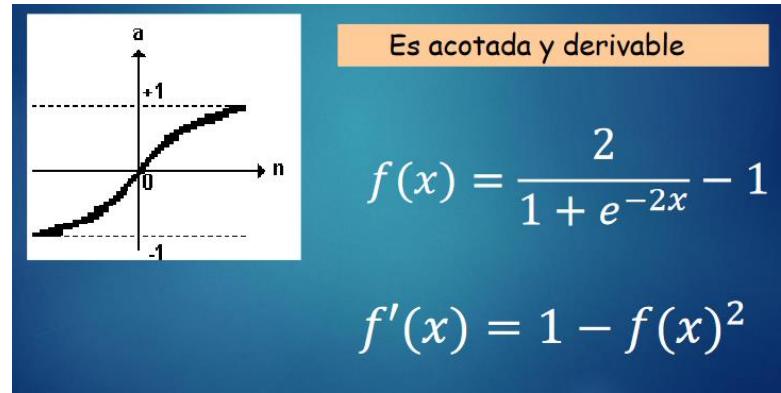
Lineal



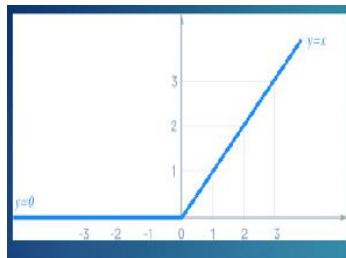
Sigmoide



Tangente (tanh)



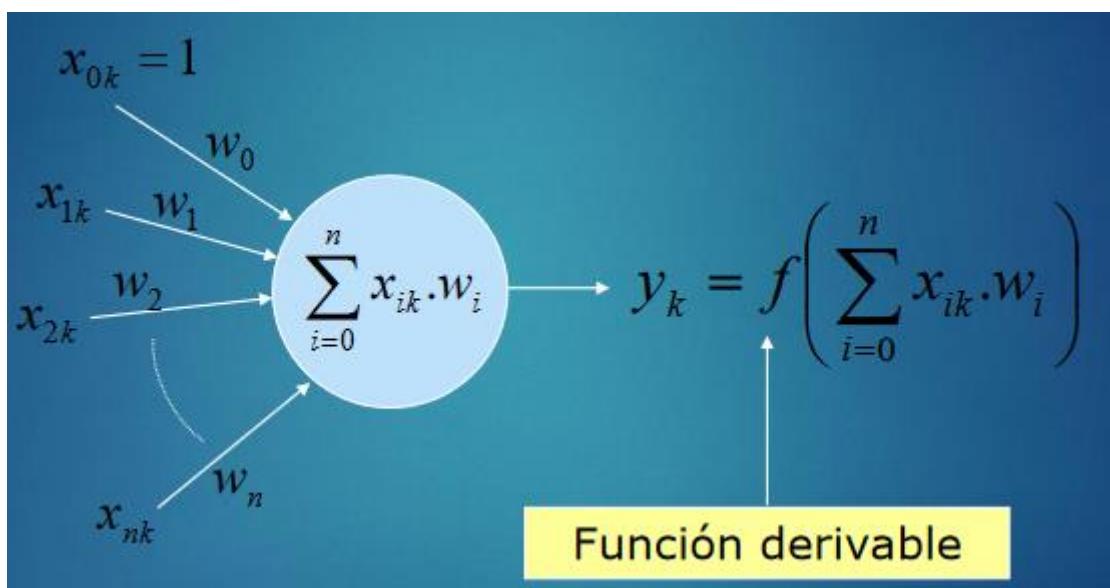
RELU (Rectified Linear Unit)



$$f(x) = \begin{cases} 0 & \text{si } x < 0 \\ x & \text{si } x \geq 0 \end{cases}$$

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

No es acotada, pero  
es "parcialmente  
derivable"



## ¿Qué busca una neurona?

La neurona intenta resolver una suerte de regresión lineal buscando el mejor hiperplano que separa mejor las clases (puede ser una recta, así como una sigmoide, dependiendo de la función de activación)

## Redes neuronales en minería de datos

- **Predicción:** La meta es obtener un modelo para poder predecir el valor de la clase dados los valores de los atributos
  - o Clasificación: usa etiquetas discretas
  - o Estimación o regresión: las clases son continuas.
- **Descripción:** Busca derivar descripciones concisas a partir de características de los datos.
- **Segmentación:** Separación de los datos en subgrupos o clases interesantes

## Redes neuronales en otros campos

En las áreas donde las redes neuronales consiguieron una supremacía por sobre otros métodos son:

- Robótica
- Simulación de eventos
- Pseudo-inteligencia artificial en juegos
- Análisis de audio, imágenes y video
- Traducciones automáticas
- Generación de texto, imágenes y video
- Manejo automático

## Aprendizaje

No es necesario tener un proceso bien definido para transformar algorítmicamente una entrada en la salida correspondiente. Sólo se necesita una colección de ejemplos representativos del problema a resolver. La RN se adapta para reproducir las salidas deseadas cuando se le presenta una entrada dada

Las redes pueden ser vistas como una función matemática muy compleja. Vista desde el punto de vista de un modelo de regresión o de clasificación, las redes pueden “aprender” a mejorar sus respuestas. En función del error, el objetivo del aprendizaje es encontrar los valores a todos los pesos de todas sus neuronas (también llamados parámetros de la red) tal que dichos valores minimicen el error cometido.

## Objetivo a minimizar

Si se posee una sola neurona, se busca minimizar

$$ECM = \frac{1}{N} \sum_{i=1}^N \left( d_i - \sum_{j=0}^M w_j x_{ij} \right)^2$$

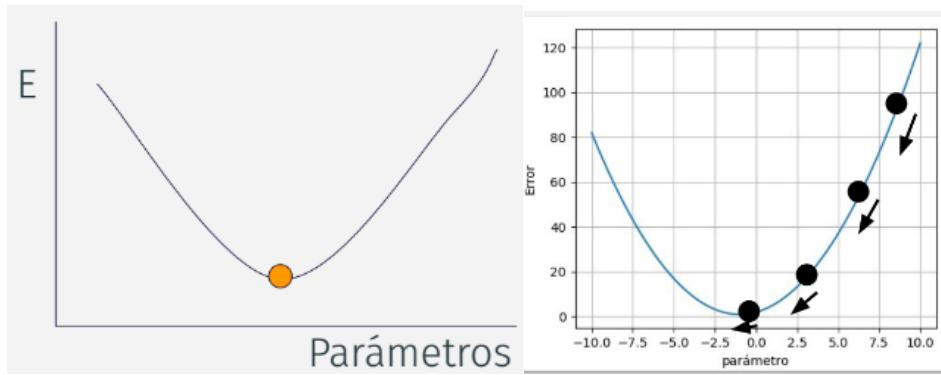
Dónde

- $d_i$  es la salida esperada
- $X_i$  es la entrada
- $W_i$  los pesos de la neurona
- N la cantidad de muestras usadas para el entrenamiento
- M la dimensión de la entrada

## Superficie del error

Función convexa

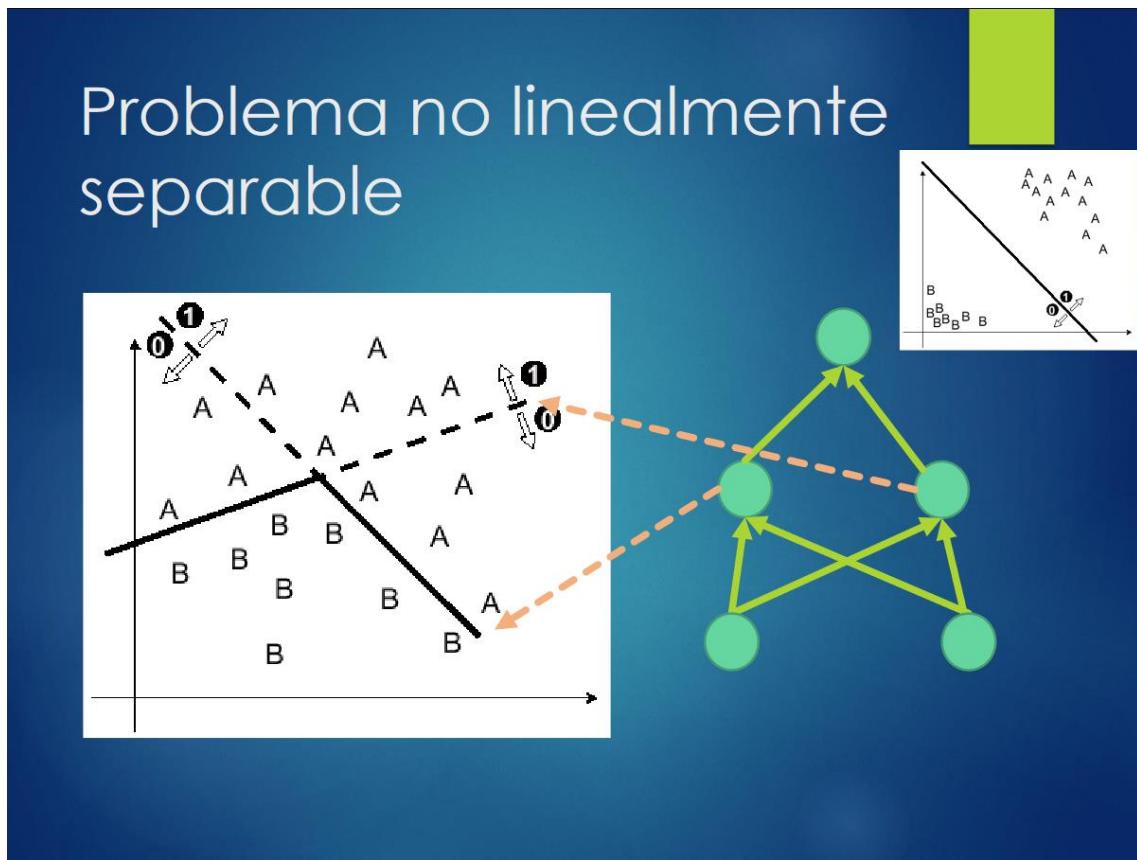
- Único mínimo (global)
- Error para regresión lineal siempre es convexa



## Problema no linealmente separable

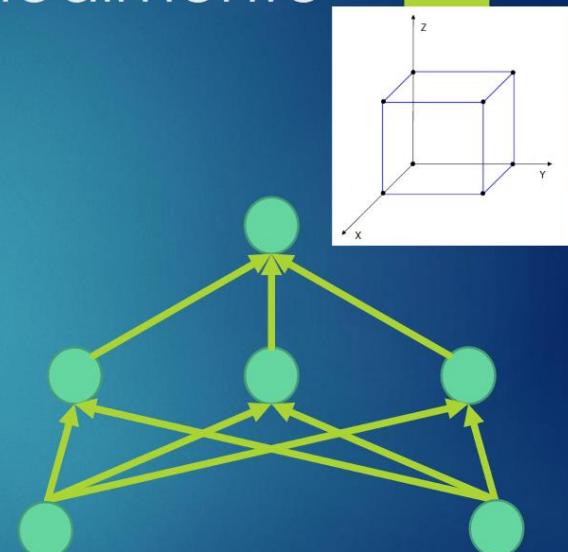
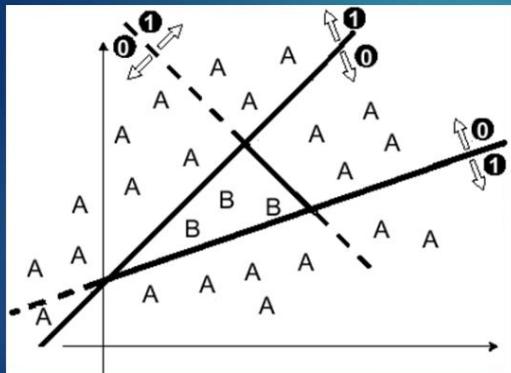
Una capa no es suficiente para resolver problemas no linealmente separables

Se necesitan muchas neuronas para resolver esto y ni siquiera, se podrían agregar más neuronas pero corréis riesgo de hacer overfitting. Entonces se puede agregar una capa oculta.



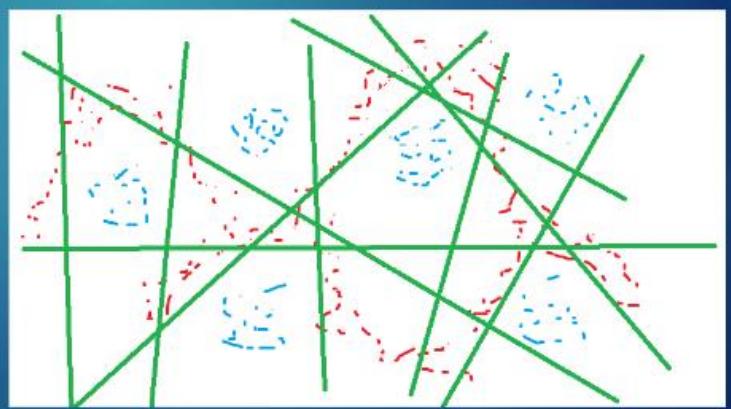
Las dos neuronas de la capa oculta transforman el espacio en linealmente separable de dos dimensiones (imagen superior derecha)

# Problema no linealmente separable



Y acá transforma el espacio en uno de tres dimensiones, por sus tres neuronas en la capa oculta.

- ▶ 10 neuronas en la capa oculta → 26 regiones
- ▶ 26 “puntos” en el espacio 10-dimencional para separar
- ▶ ¿2 clases?
- ▶ ¿7 clases?



## Red multicapa

Con el avance de la fotografía digital y las capacidades en hardware para almacenar millones de imágenes y video el interés del campo de la inteligencia artificial se volcó a reconocer e identificar patrones en imágenes.

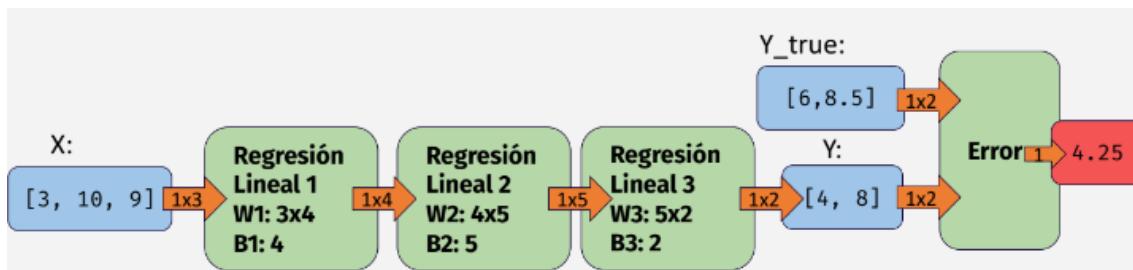
La técnica del gradiente descendente posee **limitaciones** ya que es proclive a encontrar un mínimo local (potenciado con una inicialización al azar de los pesos)

Basado en el comportamiento del cerebro humano lo que se busca es **que las primeras capas** de estas redes aprendan por si solas a detectar patrones y ejemplos similares (aprendizaje no-supervisado).

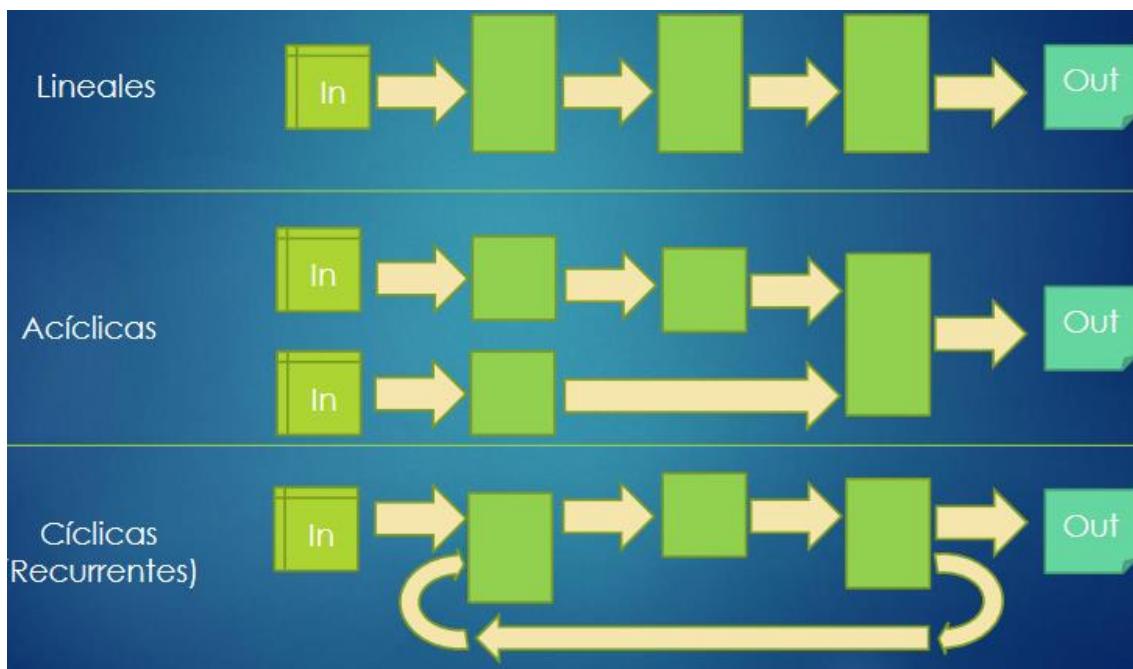
**Las últimas capas** se centran en detectar "que son" los patrones aprendidos (aprendizaje supervisado).

Función de error (una capa más):

- El cálculo del error puede ser visto como una capa más
- Capa especial
- No se usa para predecir, solo para entrenar la red



## Topología de las RNA



## Entrenamiento

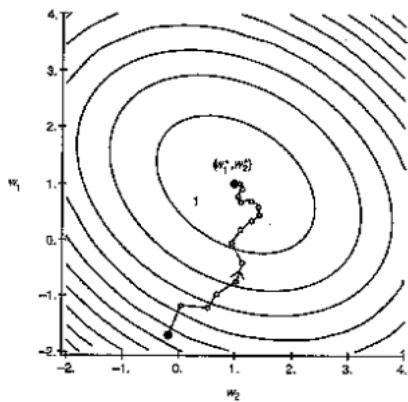
Indicar la arquitectura: la cantidad de entradas y salidas están determinadas por el **problema a resolver**.

Algoritmo de entrenamiento: forma de obtener el conocimiento que quedará almacenado en la red en los pesos de los arcos que conectan las neuronas

## Entrenar y probar.

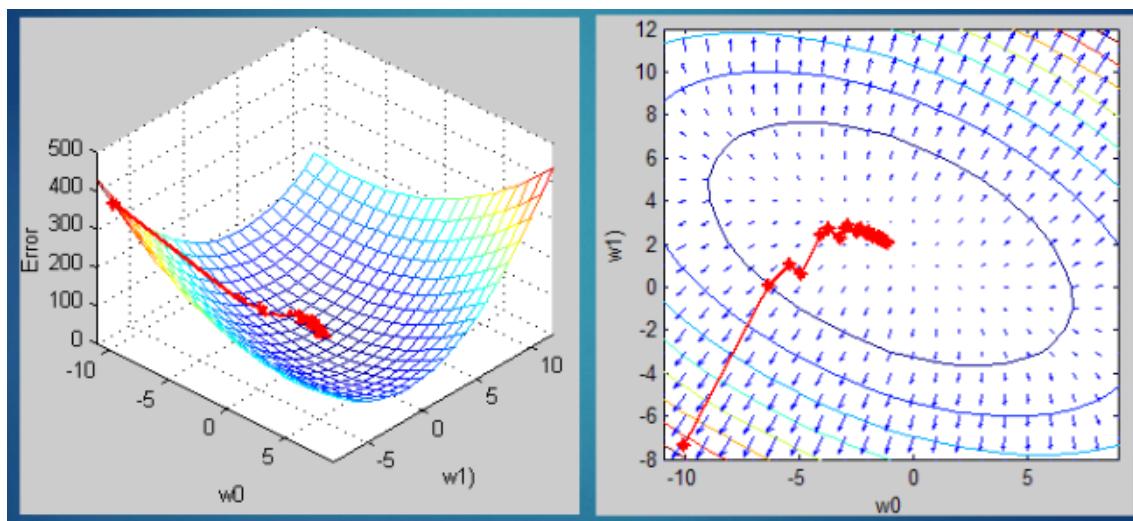
### Minimización de funciones usando el gradiente

Se busca un método de entrenamiento que, a partir de los datos de entrada, permita calcular el vector W.



Dada una función continua:

- Tomar un punto dentro del dominio de la función.
- Calcular el vector gradiente de la función en ese punto.
- Sumarle al punto anterior una fracción del gradiente negativo (para ir hacia el mínimo).
- Repetir los dos pasos anteriores hasta que la diferencia entre evaluaciones consecutivas de la función sea inferior a una cierta cota.



También se puede caer en mínimos locales

### Velocidad de aprendizaje

- Alpha demasiado chico: poco avance por iteración y alto costo computacional
- Alpha “correcto”: buen avance por iteración, razonable costo computacional.
- Alpha peligrosamente grande: “saltos” grandes, se acerca a la solución
- Alpha demasiado grande: “saltos” muy grandes, divergen

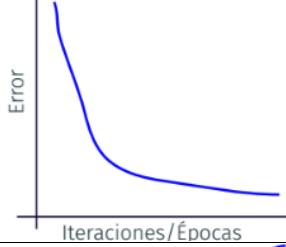
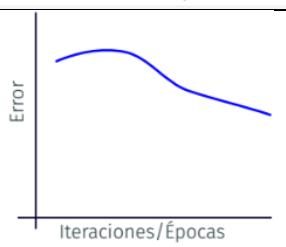
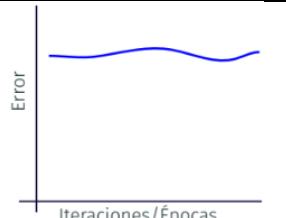
### Monitoreo del entrenamiento

¿Cuándo terminar?

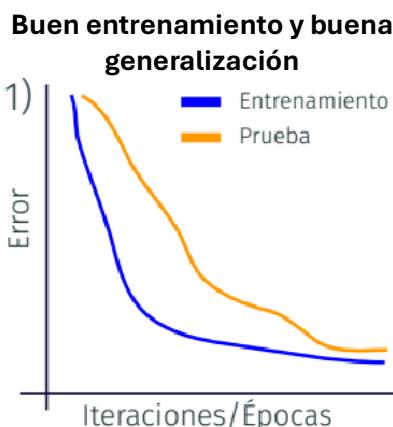
- Número de iteraciones fijas
- Tolerancia de error

¿Cómo determinar el criterio de parada?

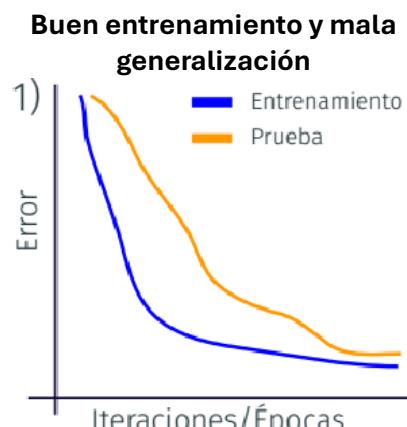
- Monitorear parámetros vs error

Convergencia rápida:	<ul style="list-style-type: none"> <li>- Ajustes menores</li> <li>- Caso típico</li> </ul> 
Divergencia	<ul style="list-style-type: none"> <li>- Alpha muy alto</li> <li>- Datos sin normalizar</li> </ul> 
Ajustes constantes	<ul style="list-style-type: none"> <li>- Faltan iteraciones</li> <li>- Alpha muy chico</li> </ul> 
Modelo no aprende	<ul style="list-style-type: none"> <li>- Problema muy complejo</li> <li>- Alpha muy chico</li> </ul> 

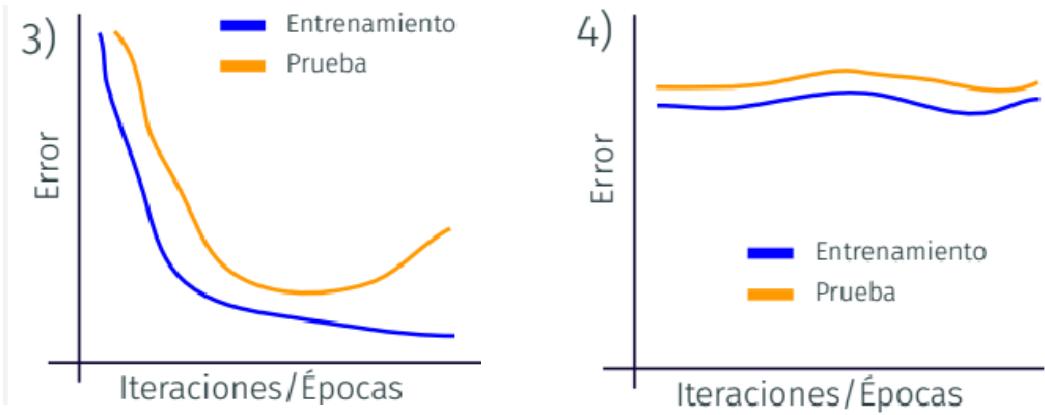
### Curvas de entrenamiento y prueba



**Buen entrenamiento, buena generalización, pero empeora a lo último (early stopping)**



**Mal entrenamiento, mala generalización: cambiar modelo**



## Algoritmo de back-propagation

Error en una neurona de la capa de salida:

$$\delta_{pk} = y_{pk} - o_{pk}$$

Donde

- $y$  es la salida deseada
- $o$  es la salida real
- $p$  se refiere al  $p$ -ésimo vector de entrada
- $k$  se refiere a la  $k$ -ésima neurona de salida

*Error cuadrático medio*

$$E_p = \frac{1}{2} \sum_{k=1}^M \delta_{pk}^2$$

$$E_p = \frac{1}{2} \sum_{k=1}^M (y_{pk} - o_{pk})^2$$

Se tomará el valor negativo del gradiente

- 1- Aplicar el vector de entrada
- 2- Calcular los valores netos de las unidades de la capa oculta
- 3- Calcular las salidas de la capa oculta
- 4- Calcular los términos de error para las unidades de salida
- 5- Calcular los términos de error para las unidades ocultas
- 6- Se actualizan los pesos de la capa de salida
- 7- Se actualizan los pesos de la capa oculta
- 8- Repetir hasta que el error resulte aceptable

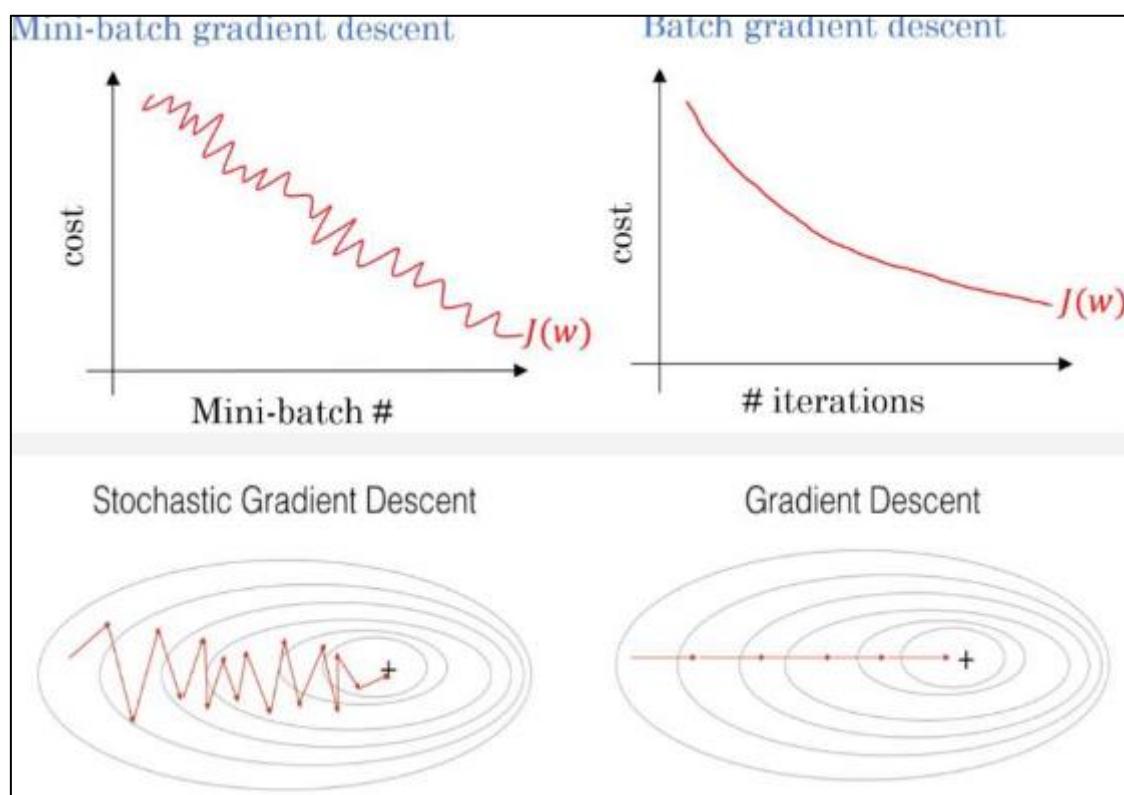
## Descenso del gradiente estocástico (entrenamiento por lotes)

Calcular el error y sus derivadas para cada muestra genera una superficie de error por “muestra” lo cual es completamente inexacto, porque está construida en función de esa sola muestra. Todo el tiempo estaría calculando superficies distintas.

En cambio, calcular todos los errores y todas las derivadas genera una superficie de error exacta, pero es computacionalmente costoso ( $O(NP)$ )

Por último, calcular el error y sus derivadas para cada “lote” genera superficies de error parciales y un gradiente ruidoso pero aproximado.

- $B$  = tamaño del lote (cantidad de muestras)
- $B$  es un hiperparámetro
  - o  $B=1$  Descenso del gradiente minibatch
  - o  $B=N$  Descenso del gradiente clásico
  - o  $B << N$  Descenso del gradiente estocástico
    - Costo de una iteración  $O(BP)$
- Épocas: cantidad de veces que se recorre el dataset completo
- Iteraciones: cantidad de veces que se actualizan los parámetros (pesos) de la red (una iteración por lote)



Cada lote tiene su superficie de error

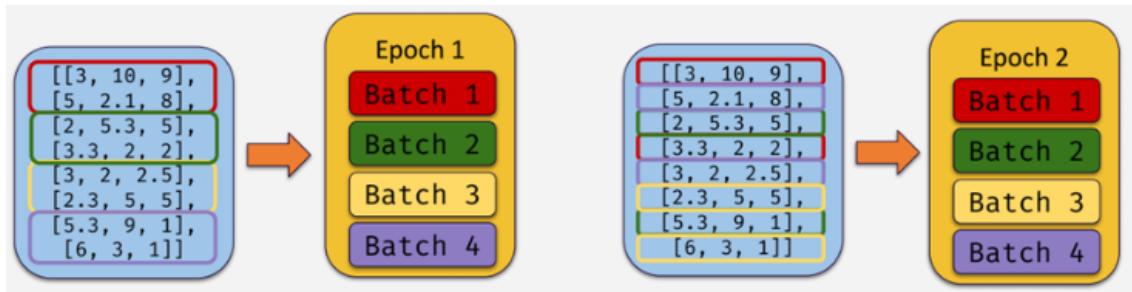
- Se redefinen las derivadas
- Equivalente, pero ruidosas

Descenso de gradiente tradicional	Descenso de gradiente Estocástico
Gradiente con <b>todos</b> los ejemplos	Gradiente con <b>lotes</b> de ejemplos
Gradientes más exactos	Gradientes ruidosos
No puede escapar de mínimos locales	Puede que escape de mínimos locales
Si $f$ convexa, garantiza mínimo global	Mínimo global no garantizado
Lento, poco escalable	Rápido, escalable

## Épocas, lotes e iteraciones

### Rearmando lotes

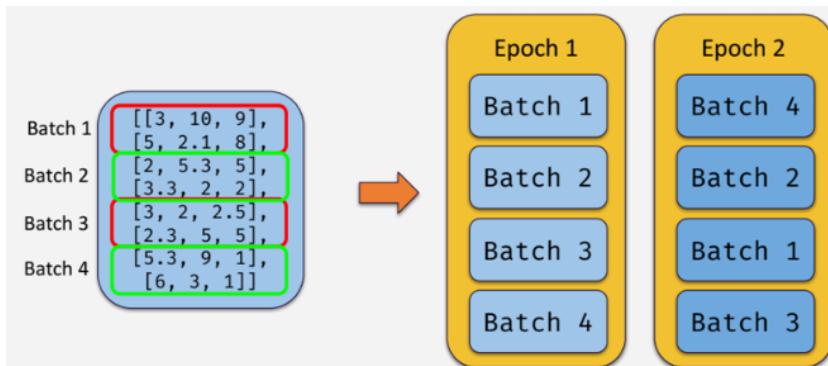
- Ayuda al entrenamiento
  - o Genera ruido
  - o Remueve efectos de orden de las muestras



### Reordenando lotes

Mismo objetivo que rearmar lotes

- Mismo efecto



## Dificultades

- Si son mal entrenadas pueden resultar en un modelo con overfitting
  - o Arquitectura incorrecta
  - o Tiempo de entrenamiento insuficiente
  - o Datos de entrenamiento inadecuados
- Tienen limitaciones en su capacidad de aprendizaje
- No siempre es posible encontrar la arquitectura adecuada
- Poseen el inconveniente del sobreajuste
- Son proclives a ser “engañadas cayendo en mínimos locales”

- Generalmente poseen fuerte dependencia con los valores iniciales en los pesos escogidos aleatoriamente.
- Frecuentemente el entrenamiento requiere mucho tiempo de procesamiento

## Capacidad de generalización

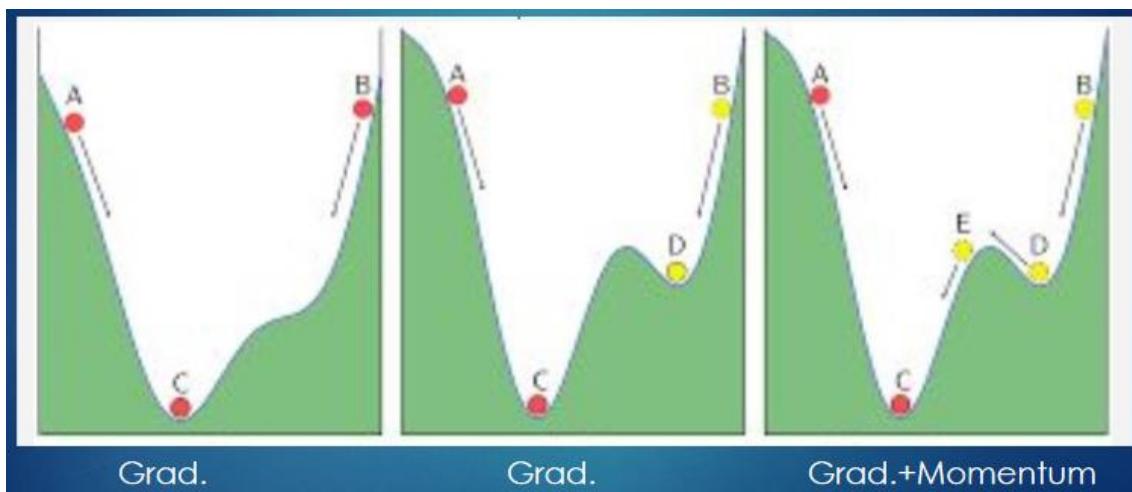
A mayor cantidad de neuronas en la capa oculta, la red puede variar más rápido en respuesta a los cambios de entrada.

### ¿Mínimo local o global?

El problema se resuelve utilizando la dirección del gradiente junto con un componente aleatorio que permita salir de los mínimos locales (subiendo en lugar de bajar)

### Término de momento

- Alpha maneja la velocidad de aprendizaje.
- Si su valor se incrementa demasiado, la red puede desestabilizarse.
- Una forma de solucionar esto es incorporar, a la modificación de los pesos, un término que incluya una proporción del último cambio realizado. Este término se denomina momento
- Permite saltar mínimos locales espurios o “malos”



## Regresión múltiple (una neurona de salida no es suficiente)

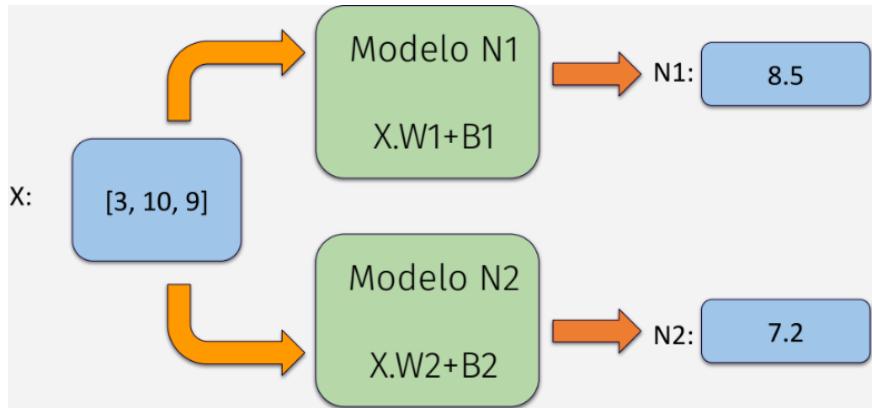
- Predecir M valores a partir de N valores (variables de entrada)
- Puede usarse para
  - Transformar un vector en otro
  - Predecir la posición de un objeto a partir de una imagen
  - Generar un texto a partir de un sonido
  - Generar una imagen a partir de un vector de características
  - Tareas de clasificación múltiples clases

Ejemplo: predecir la **posición** y **tamaño** de un cartel

- Vector de 4 valores (x,y, alto, ancho)

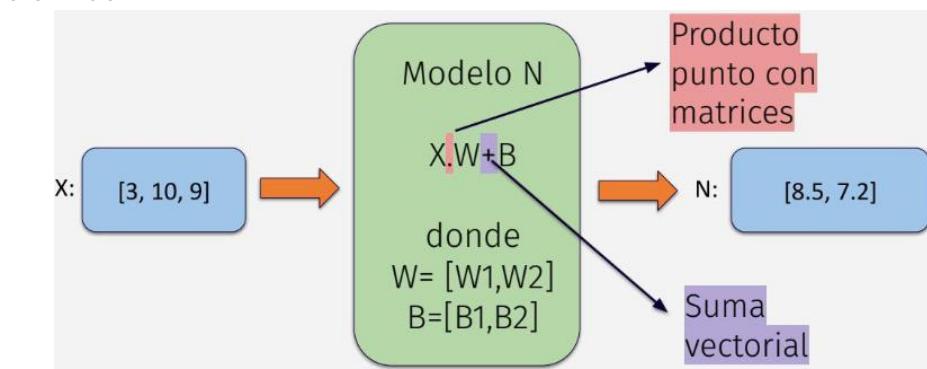
### Versión simple

Un modelo para cada variable de salida (1 para alto, 1 para ancho, etc)



Combinar los parámetros en

- Matriz W de 3x2
- Vector B de 2x1



Determinar el tamaño de cada capa

Tener una neurona de salida por cada clase esperada.

La idea es que solo una se “encienda”, ej: [0,0,1,0,0]

## Salida de la red

Convertir cada clase esperada a un vector binario (codificación One-Hot)

Clase A	→	0	0	0	0	1
Clase B	→	0	0	0	1	0
Clase C	→	0	0	1	0	0
Clase D	→	0	1	0	0	0
Clase E	→	1	0	0	0	0

Ojo: depende de la función de transferencia utilizada, puede ser 0 o -1.

Es improbable que en la salida solo haya unos y ceros

- 0.02, 0.09, 0.12, 0.97, 0.003

Redondeando tenemos 0, 0, 0, 1, 0

Que es el representante de la Clase B

¿Qué pasa si en la salida tenemos...?: 0.02, 0.93, 0.12, 0.97, 0.83

Redondeando tenemos 0, 1, 0, 1, 1

Que no representa ninguna clase

- Podemos decir que el ejemplo es clasificado como “desconocido”

¿Y qué pasa si en la salida tenemos...? 0.42, 0.53, 0.48, 0.47, 0.43

Redondeando tenemos 0, 1, 0, 0, 0 **¿es válido?**

**Alternativa:** aplicar un umbral y redondear

- Ej: si es menor a 0.2 se redondea a 0
- Ej: si es mayor a 0.8 se redondea a 1
- En otro caso la red responde como “desconocido”
- Ojo: depende de la función de transferencia utilizada, puede ser 0 o -1

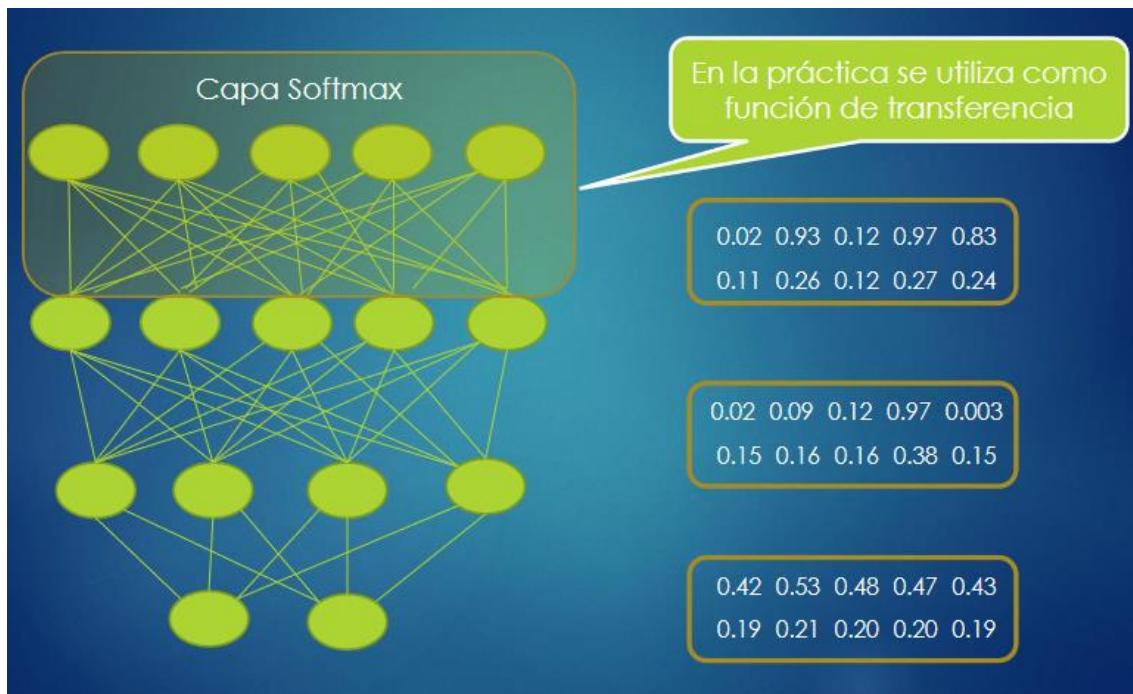
## Capa softmax

El uso del error cuadrático como medida de error tiene algunos inconvenientes y si estamos ante un problema de clasificación y queremos estimar la probabilidad de cada clase, sabemos que la suma de las salidas debería ser 1, pero no estamos usando esa información para entrenar la red neuronal.

La función softmax fuerza que las salidas de la red representen una distribución de probabilidad:

$$\sigma(z)_j = \frac{e^{z'_j}}{\sum_{k=1}^K e^{z'_k}}$$

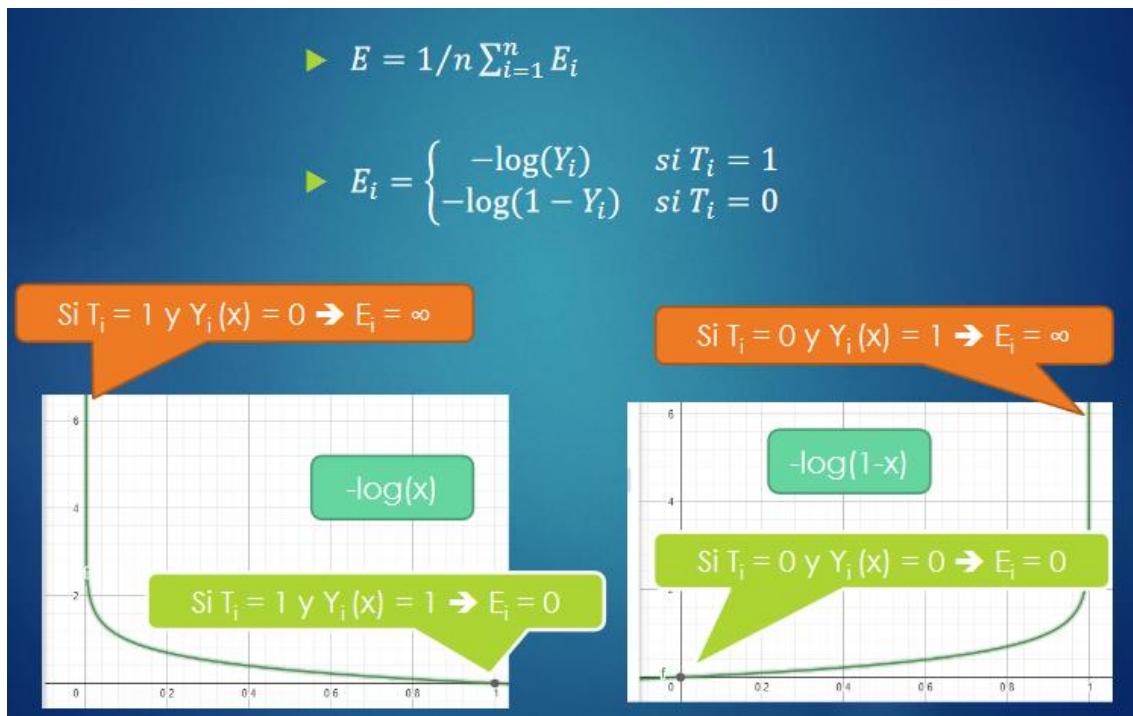
Donde Z es un vector k-dimensional de valores reales y  $\sigma$  es un vector k-dimensional donde todos sus elementos son valores entre 0 y 1 y la suma de ellos es 1.



## Entropía cruzada (obteniendo probabilidades)

El error cuadrático medio no es convexo

En cambio, la entropía cruzada sí es convexa para  $E_i$



## Función de error

Cambia la función de salida

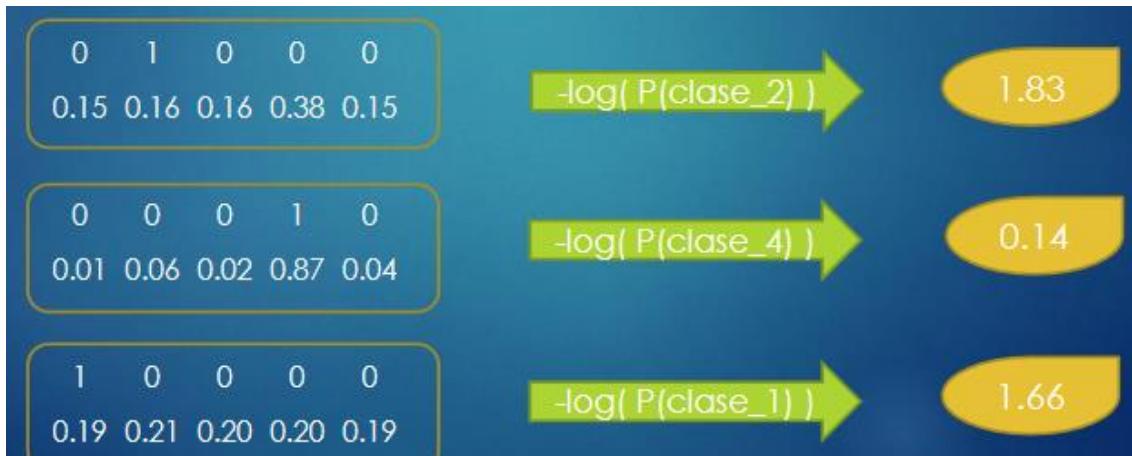
- Cambia la función de error

Salida: softmax

- Error: entropía cruzada

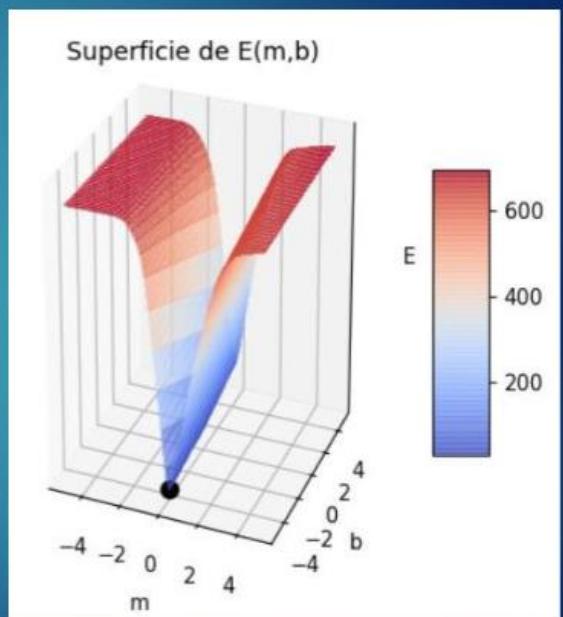
Penaliza la salida de la clase verdadera

- $-\log(P(\text{clase\_verdadera}))$



►  $E_i = -\log(Y_i)$

- Las derivadas son fáciles de calcular
- Se utiliza la técnica del gradiente para la actualización de los pesos de la red.

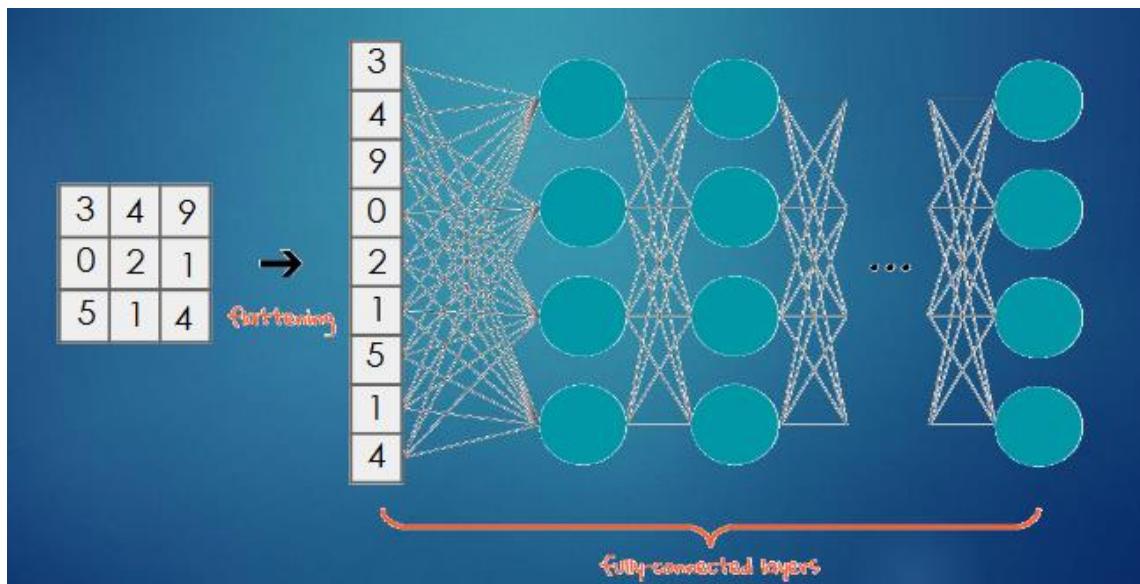


## Redes convolucionales

En una imagen en color cada pixel tiene tres componentes, con distintas representaciones:

- RGB
- HSV
- HSL
- RYB
- CMYK

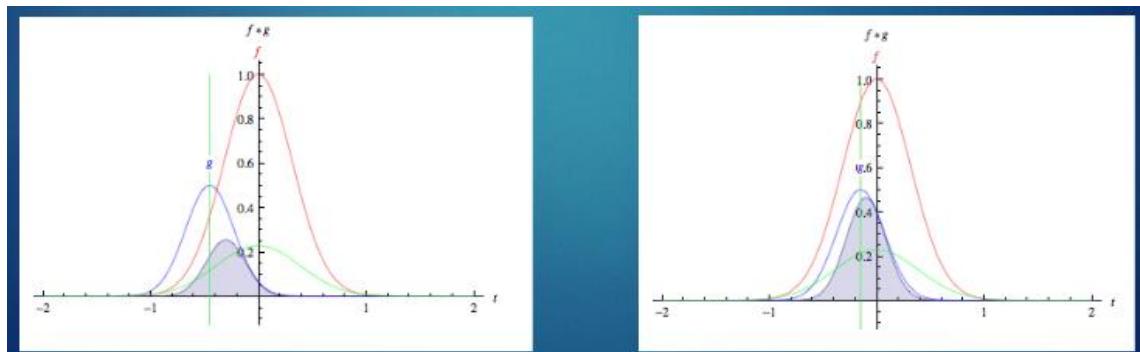
Las imágenes tienen los pixeles en formato matricial (o tensorial). Un modelo neuronal no acepta esto, por ello es necesario "aplanar" la imagen.



## Convolución: Haciendo pasar una función sobre otra

En matemática la convolución entre dos funciones  $f$  y  $g$  es una medida de cuantos se "solapan" cuando una pasa sobre la otra.

El producto entre ambas es la "mezcla" de ambas funciones. Su punto máximo está dado por los máximos de ambas funciones.

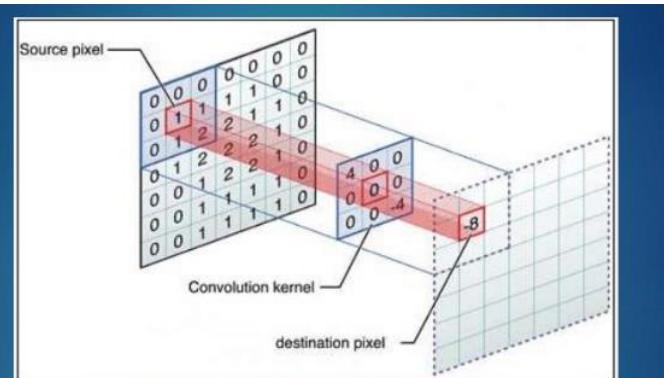


## Convolución discreta en 1D

$x$	5	0	2	-1	3	0	2
$f$	-1	0	1				
$y$	-3	-1	1	1	-1		

$$y_i = \sum_{k=1}^3 x_{i+k-1} * f_k$$

## Convolución discreta en 2D



$$y_{i,j} = \sum_{k=1}^3 \sum_{l=1}^3 x_{i+k-1, j+l-1} * f_{k,l}$$

## Filtros convolucionales

En el procesamiento digital de imágenes un filtro convolucional es una matriz (por lo general cuadrada) de valores

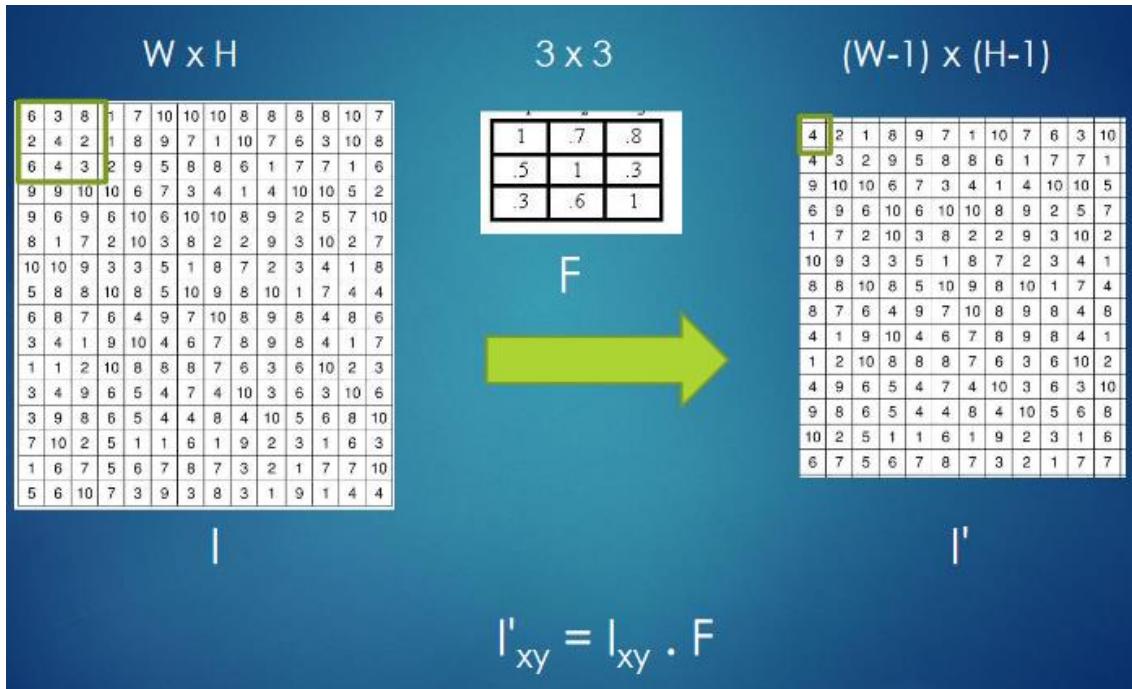
Los filtros se utilizan para obtener una imagen resultado con las características más relevantes para tareas como clasificación o búsqueda de patrones.

- Filtros para la detección de bordes



- Filtros para la erosión o dilatación





## Hiperparámetros

- Stride: cuantos pasos se mueve el filtro con cada iteración
- Padding: relleno de los bordes para poder hacer la convolución

## Redes convolucionales

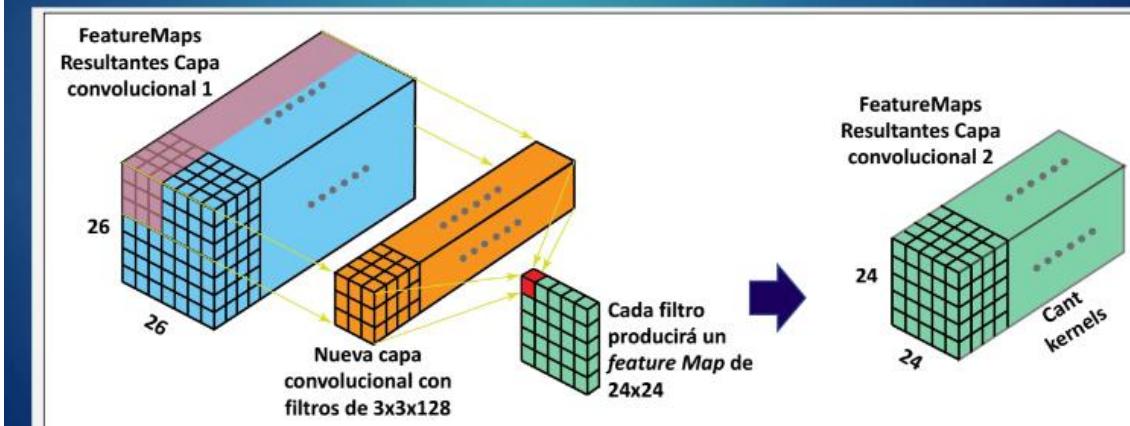
- Las redes convolucionales están inspiradas en la estructura del **sistema visual**.
- Tienen un amplio uso en el reconocimiento de imágenes.
- Las redes convolucionales por lo general están formadas por cinco o más capas.
- Estas capas se dividen en:
  - o Capas convolucionales
  - o Capas de subsampling

## Capa convolucional

- Consiste en aplicar **uno o más filtros** (denominados kernels) a la imagen de entrada.
- El resultado de aplicar un kernel obtiene un único valor (un pixel)
- A la matriz (o tensor) resultante se lo conoce como mapa de activación (feature map).
- El ancho y alto del feature map resultante dependerá del paso con el cual se hace desplazar los kernels por la imagen de entrada.
- La profundidad del feature map dependerá de la cantidad de kernels que se utilizan en la capa convolucional.

# Capa convolucional

- ▶ Feature map de entrada → (26 26 x 128)
- ▶ 54 kernels de (3 x 3)
- ▶ Feature map de salida → (24 x 24 x 54)



- Los valores de los filtros son los pesos de la red
  - La red deberá aprender o encontrar los valores de los pesos de cada kernel que permitan identificar una imagen con el menor error posible
- 
- Feed forward con 64 neuronas ocultas
    - o  $212 \times 220 \times 3 = 139920$  entradas
    - o  $(139920 + 1) \times 64 = 8.954.944$  pesos
  - Red convolucional con 64 filtros
    - o  $64 \times 3 \times 3 \times 3 + 64$  bias = 1792 pesos

Se busca que cada neurona busque una característica por si sola

- Se aplican varias conjuntos de filtros
- La red aprenderá de todos ellos a clasificar imágenes o detectar patrones
- Un filtro que "barra" la imagen en r filas y c columnas dará como resultado una matriz de r x c. Si al mismo tiempo aplicamos m filtros distintos obtendremos un volumen de salida de r x c x m.

Ventajas y desventajas

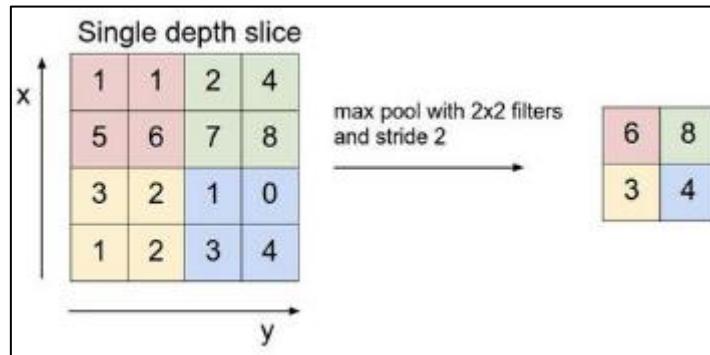
- Son tolerantes a la translación
- La rotación de una imagen podría hacer que no se encuentren los patrones buscados

## Capa subsampling (downsampling o pooling)

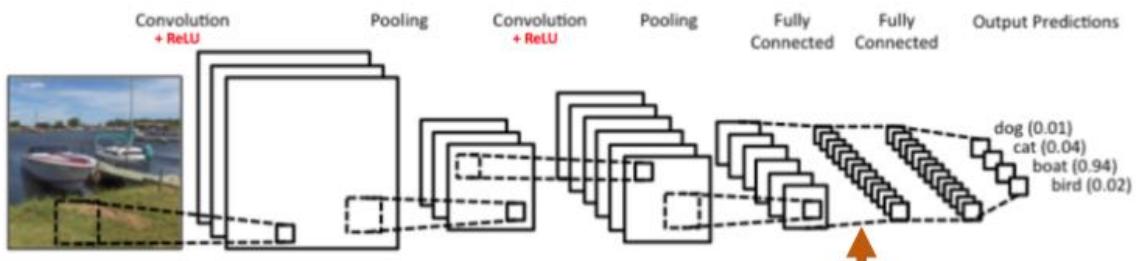
Se encargan de reducir el tamaño especial del feature map.

Se define un tamaño de división para la imagen. De cada porción de la imagen se toma la característica más importante.

Por lo general la característica más importante es el valor máximo



Arquitectura completa de una red convolucional con dos capas convolucionales y tres capas densas



## Características de la red

El entrenamiento de una red convolucional consiste en encontrar los filtros que permiten clasificar correctamente la imagen

- El aprendizaje es supervisado, similar al algoritmo de backpropagation.
  - o Agrega variantes de como calcular la derivada del error en las capas de pool.
- Los resultados dependerán de los hiperparámetros del algoritmo
  - o Cantidad de filtros (depth)
  - o Cantidad de pasos al mover el filtro (stride)
  - o Rellenado de ceros (zero padding)
  - o Tamaño de filtro

## Resumen

Las Capas convolucionales 2D tienen filtros (kernels) que se entrena para detectar diferentes características.

Cada filtro genera un Feature Map de NxNx1. Donde N dependerá del tamaño original de la imagen, del filtro, el padding y el stride que usemos.

Todos los Feature Maps de los distintos filtros de una capa convolucional se apilan, generando una nueva “imagen” (Activation Map) de  $N \times N \times F$ , siendo  $F$  la cantidad de filtros.

Las capas Pooling permiten reducir la dimensionalidad del problema, haciendo no solo más rápido el entrenamiento sino más eficaz al momento de generalizar.

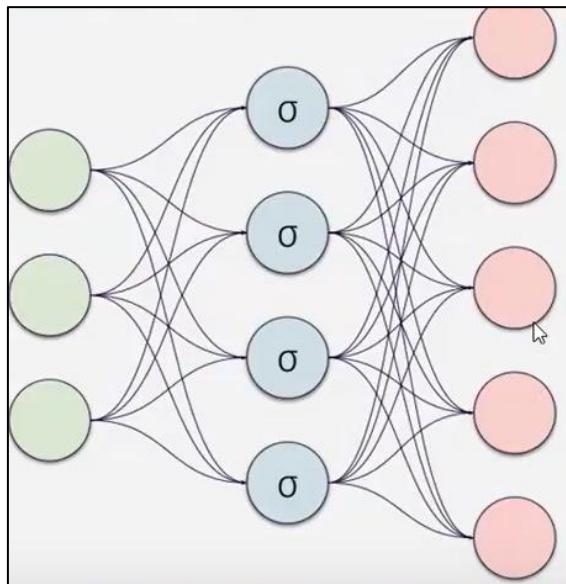
El modo más ordenado de realizar una arquitectura ConvNet es intercalar capas Convolucionales con capas Pooling hasta llegar a las capas Dense (Feed-Forward) que discriminarán las características aprendidas por las capas anteriores.

## Aproximador universal

### El teorema de Cybenko

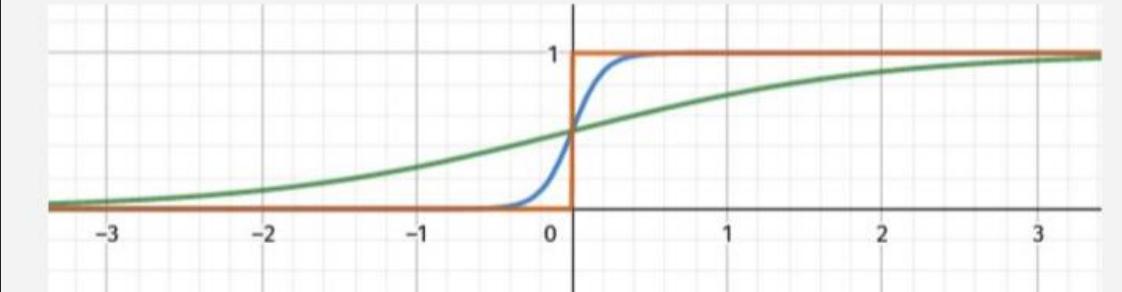
Una red de dos capas permite aproximar cualquier función (aproximador universal)

- Aumentando el número de parámetros (neuronas) decrementa el error de manera arbitraria



## Función sigmoidea

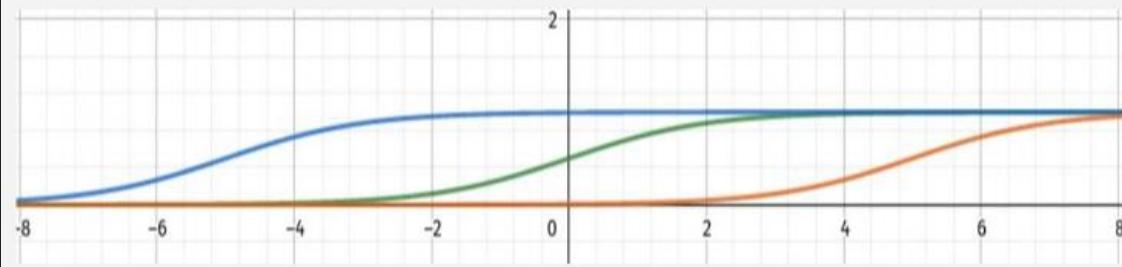
- Sigmoidea
  - $\sigma(x) = 1 / (1+e^{-x})$
  - $\sigma(x) = 1 / (1+e^{-(wx+b)})$
- Variando w y b
  - Cambia salida
- w varia pendiente
  - $\sigma(x)$
  - $\sigma(10x)$
  - $\sigma(1000x)$  (función step)



Converge rápido a cero a medida que vamos a menos infinito y a 1 cuando va a +infinito

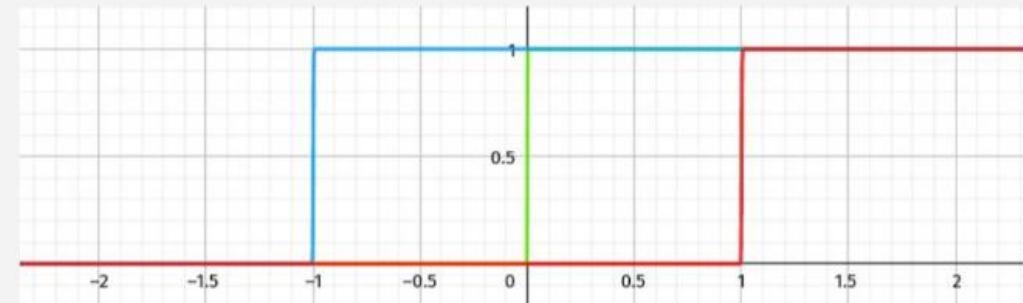
Con el b corremos el gráfico

- Sigmoidea
  - $\sigma(x) = 1 / (1+e^{-x})$
  - $\sigma(x) = 1 / (1+e^{-(wx+b)})$
- Variando w y b
  - Cambia salida
- b varia posición
  - $\sigma(x)$
  - $\sigma(x+10)$
  - $\sigma(x-10)$



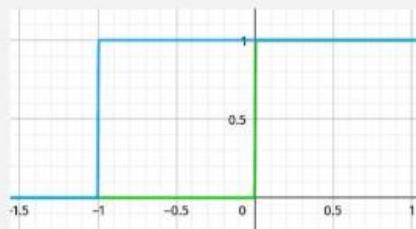
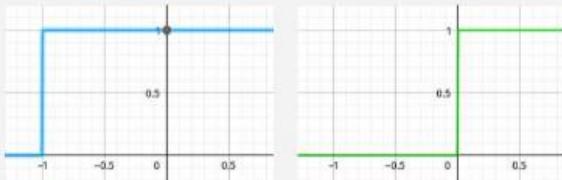
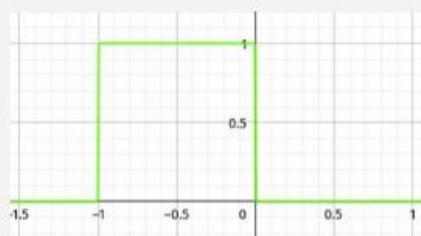
y con el w cambiamos la inclinación

- Sigmoidea
  - $\sigma(x) = 1 / (1+e^{-x})$
  - $\sigma(x) = 1 / (1+e^{-(wx+b)})$
- Variando w y b
  - Cambia salida
- w y b combinadas
  - $\sigma(1000x)$
  - $\sigma(1000x+1000)$
  - $\sigma(1000x-1000)$

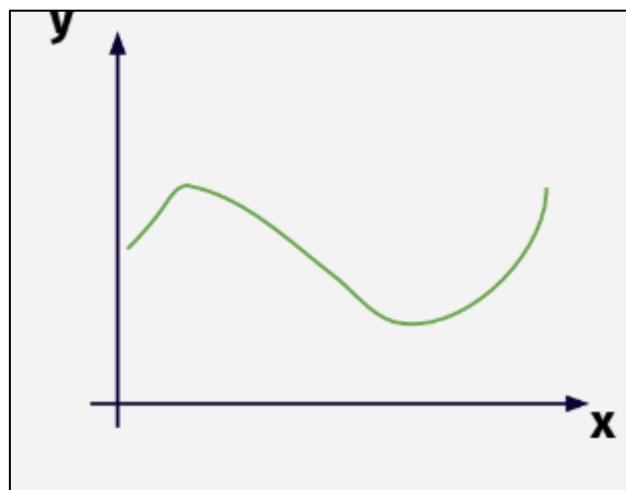


Combinando sigmoids obtengo un escalón:

- Sigmoidea
  - $\sigma(x) = 1 / (1+e^{-x})$
- Selector de región
  - $\sigma(1000x)$
  - $\sigma(1000x+1000)$
  - $\sigma(1000x+1000)-\sigma(1000x)$



Problema de regresión:  $Y = f(x)$

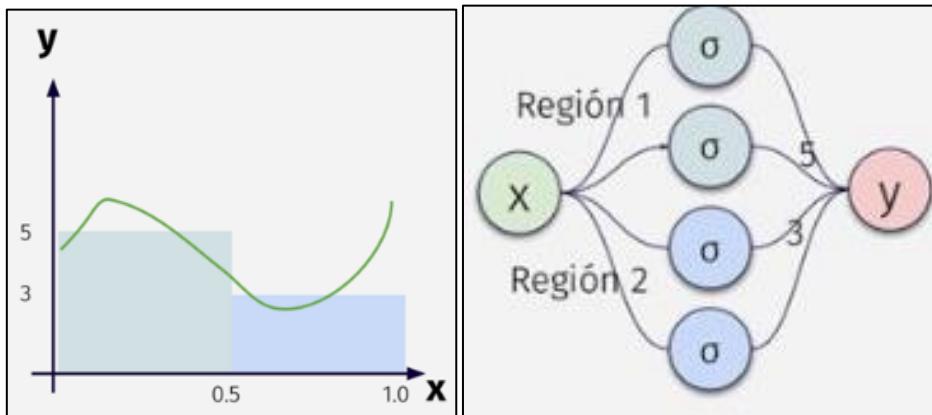


$$\text{Red: } Y = W_2 * \text{sig}(w_1 * x + b_1) + b_2$$

Estrategia: crear selectores de región para diferentes rangos de x

Ejemplo con cuatro neuronas ocultas:

- Dos neuronas por región
  - o Región 1: [0,0.5]
    - Ponderador: 5
  - o Región 2: [0.5,1]
    - Ponderador: 3

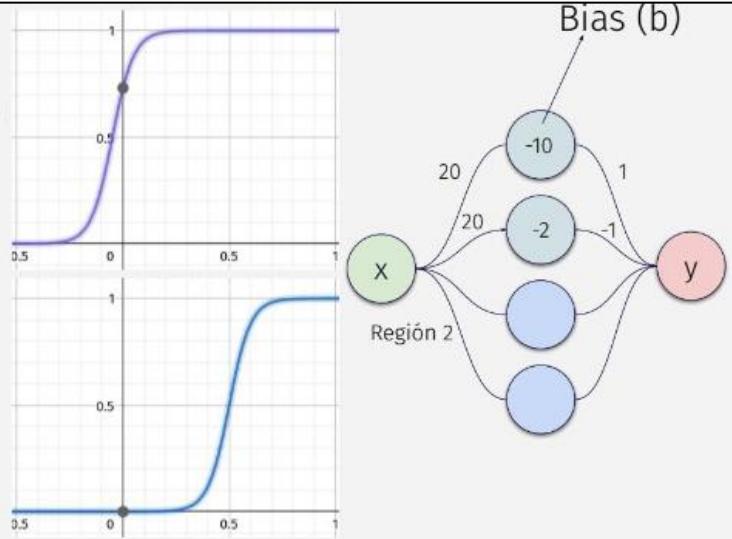
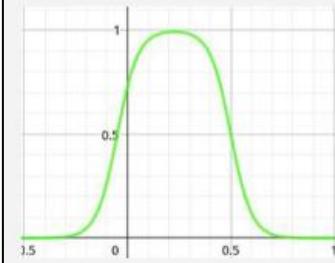


Si logro encontrar dos sigmoideas con diferentes escalones logro una aproximación (en este caso es muy mala)

A estas regiones se las conocen como selectores:

### Selector de región 1

- Sigmoidea
  - o  $\sigma(x) = 1 / (1+e^{-x})$
  - o  $\sigma(x) = 1 / (1+e^{-(wx+b)})$
- $g(x) = \sigma(20x-10)$
- $h(x) = \sigma(20x-2)$
- $g(x)-h(x)$

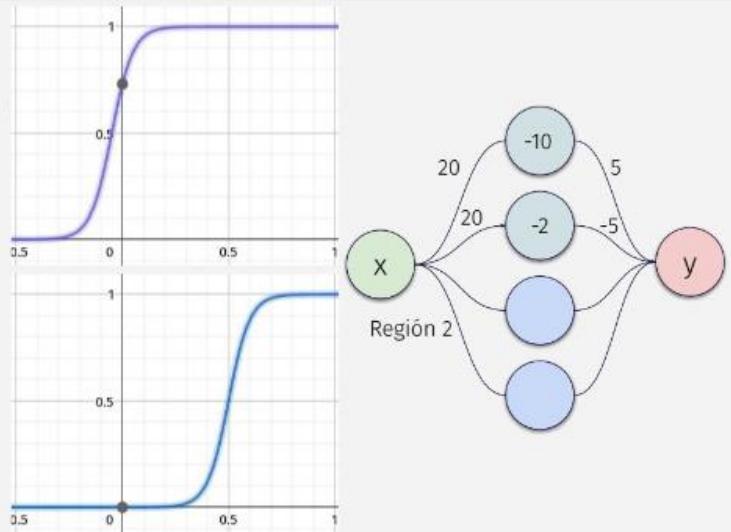
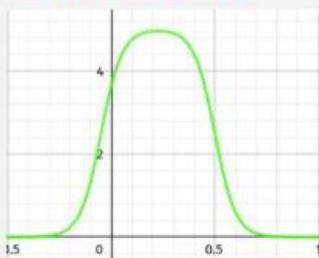


El arco de cada sigmoidea va a ser el peso w y el bias va a estar metido en la neurona.

Sigue funcionando de la misma manera pero cambia la función de transferencia (en vez de ser escalones son dos sigmoideas), por eso queda más suavizado.

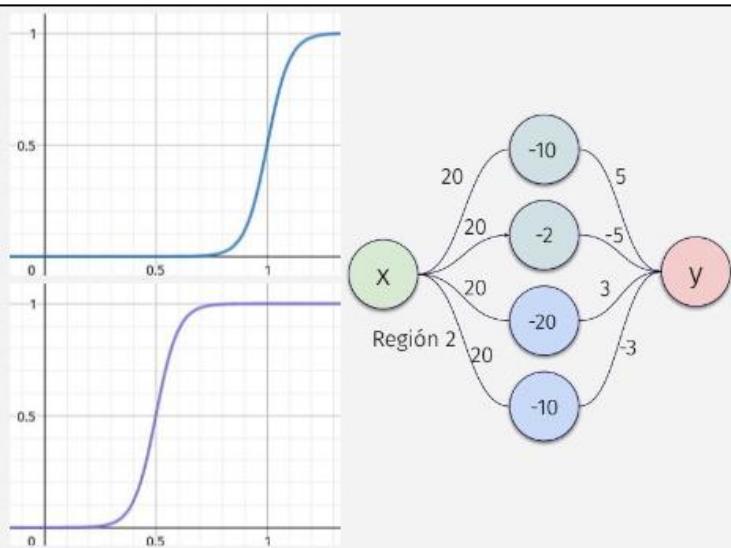
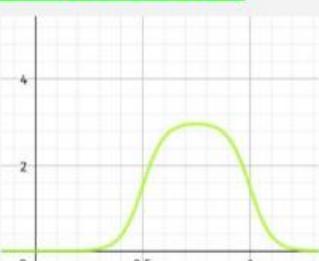
Pero está acotado entre cero y uno, ¿cómo hago para que llegue al 5? Multiplico ambas sigmoideas por 5.

- Sigmoidea
  - $\sigma(x) = 1 / (1+e^{-x})$
  - $\sigma(x) = 1 / (1+e^{-(wx+b)})$
- $g(x) = \sigma(20x-10)$
- $h(x) = \sigma(20x-2)$
- **5 g(x) - 5 h(x)**



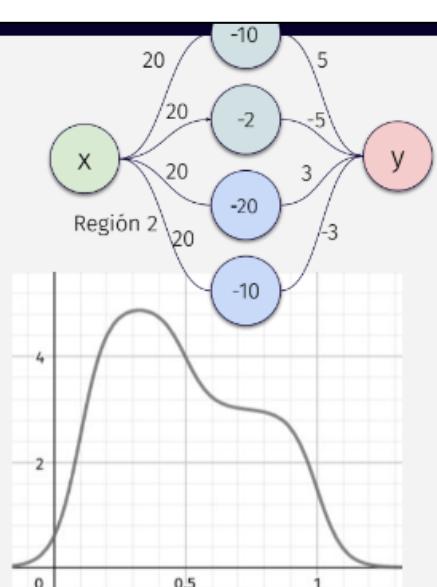
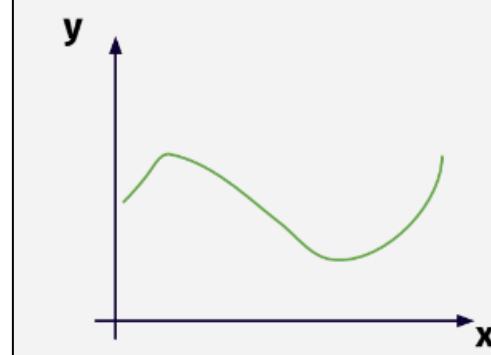
Las otras dos neuronas harán lo mismo, pero multiplicadas por 3

- Sigmoidea
  - $\sigma(x) = 1 / (1+e^{-x})$
  - $\sigma(x) = 1 / (1+e^{-(wx+b)})$
- $g(x) = \sigma(20x-20)$
- $h(x) = \sigma(20x-10)$
- **3 g(x) - 3 h(x)**



Cuando sumo las cuatro neuronas obtengo el aproximador:

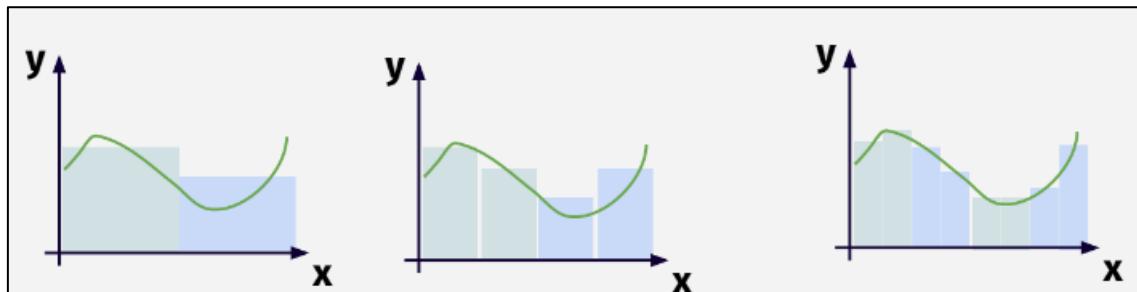
- $y = 5 \sigma(20x-10) - 5 \sigma(20x-2) + 3 \sigma(20x-20) - 3 \sigma(20x-10)$



Sigue sin parecerse, pero se asemeja más. Estos valores están puestos a mano, pero la idea es entrenando, si bien solo hay que entrenar una neurona de cada región y que la otra sea su complemento (en este ejemplo, que también esté multiplicado por 5 o 3).

## Aproximador con más funciones

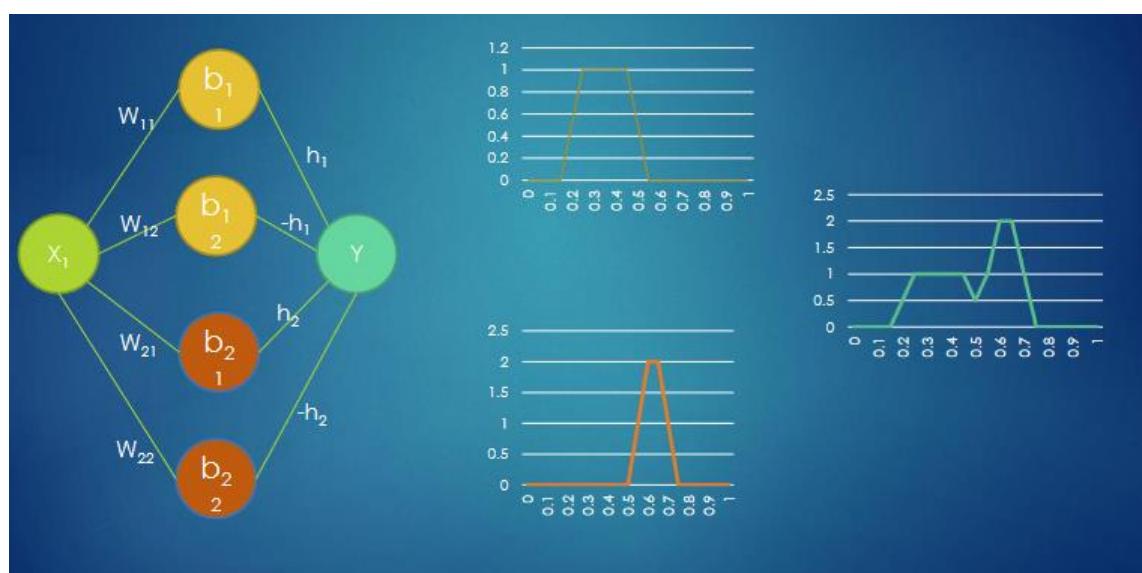
Más regiones (más parámetros) ¿Cuántos selectores? Si son muchos vas a sobreajustar.



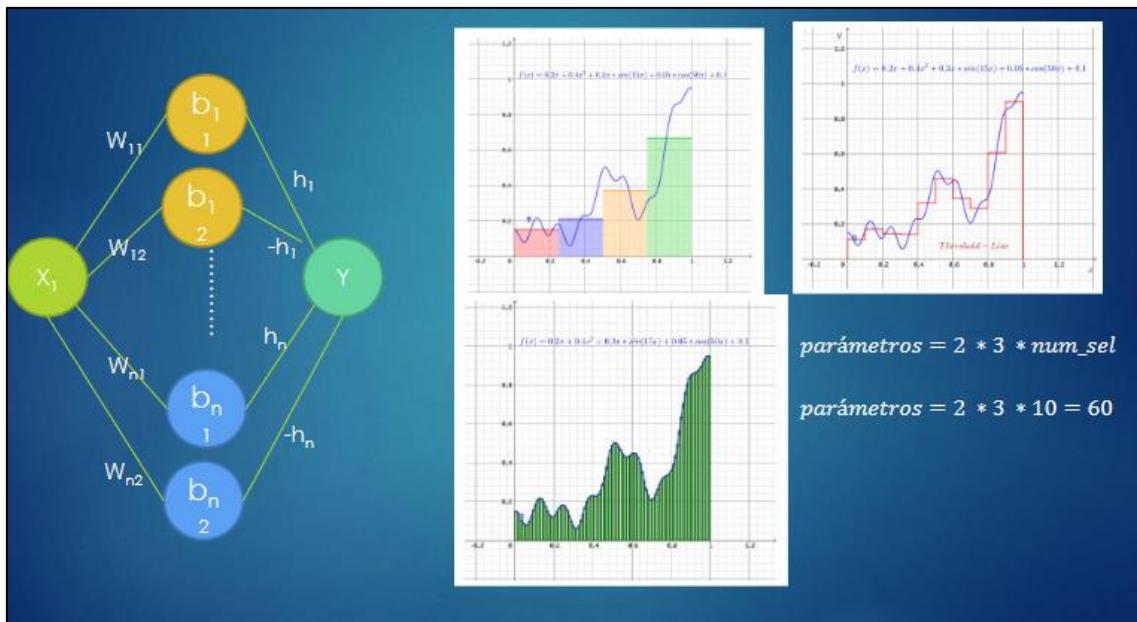
Una variable de entrada, un selector de regió



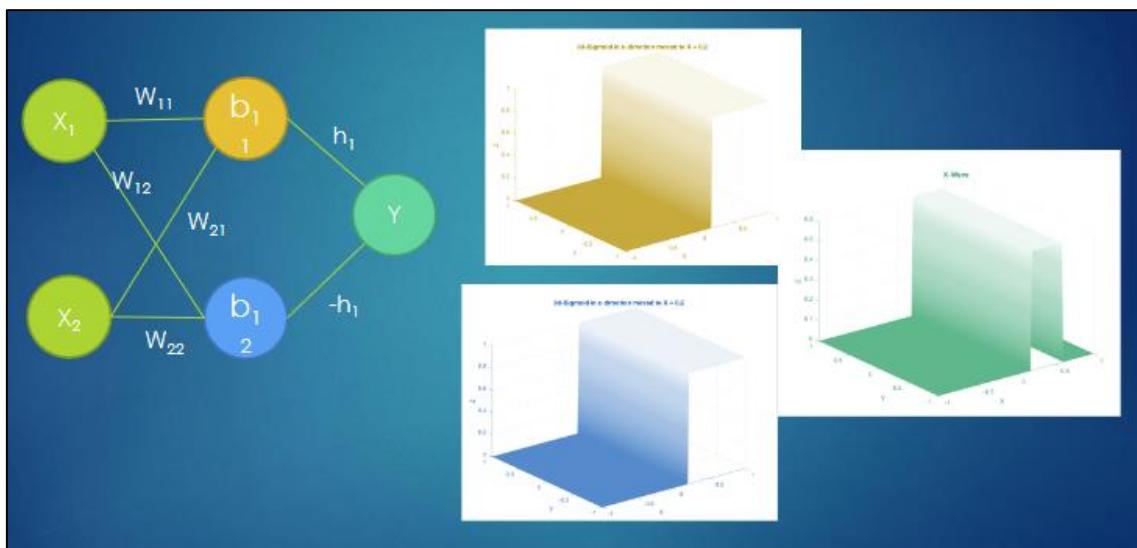
Una variable de entrada, dos selectores de región



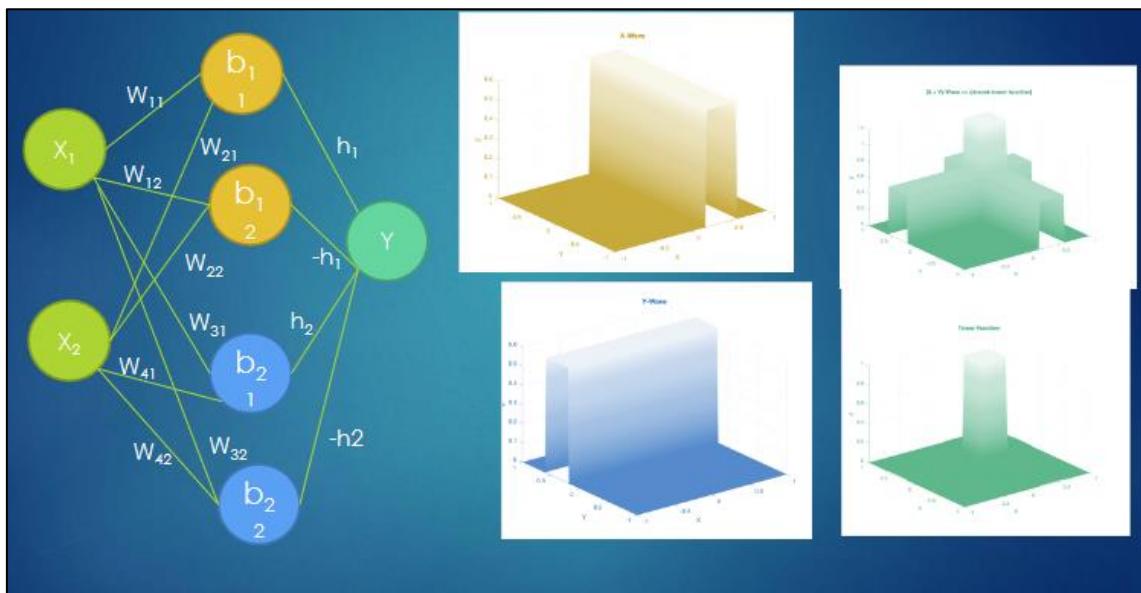
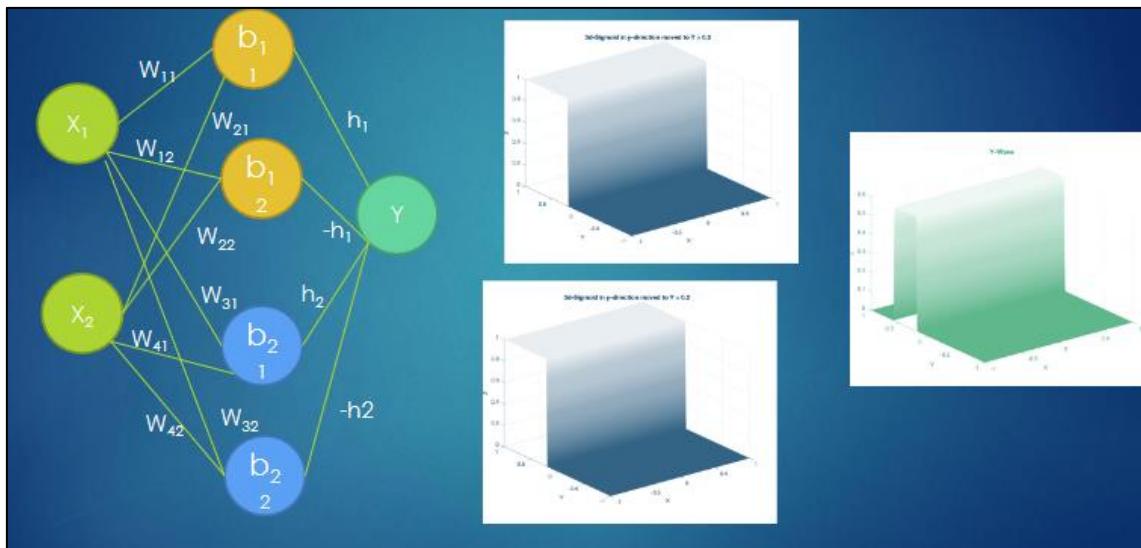
## Una variable de entrada ¿cuántos selectores de región?



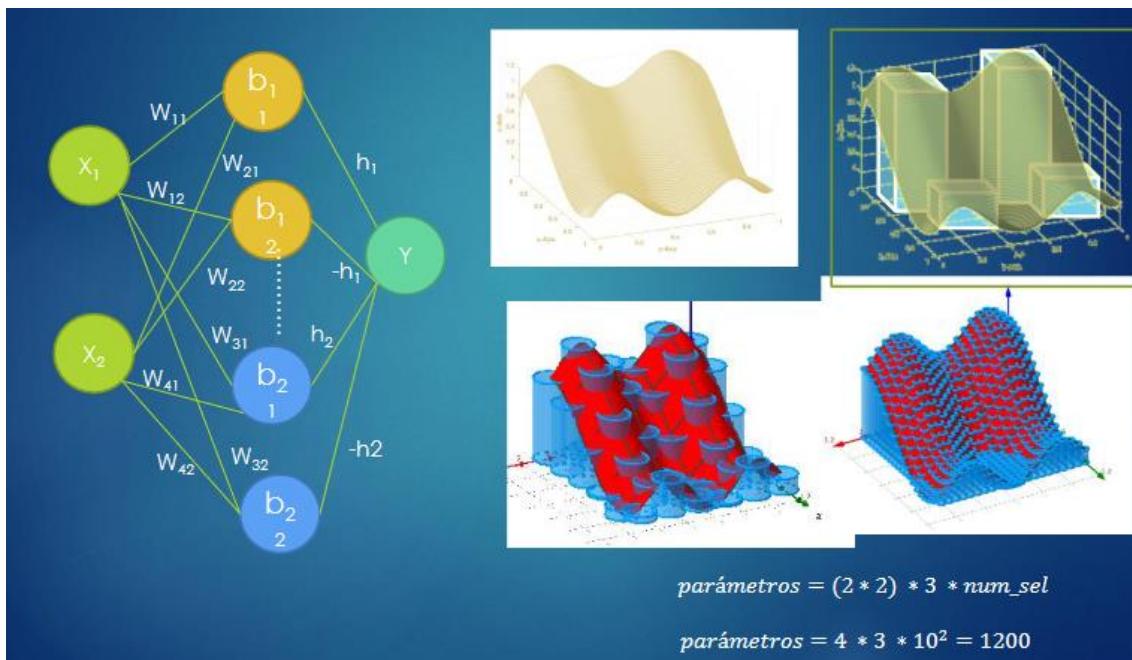
## Dos variables de entrada, un selector de región



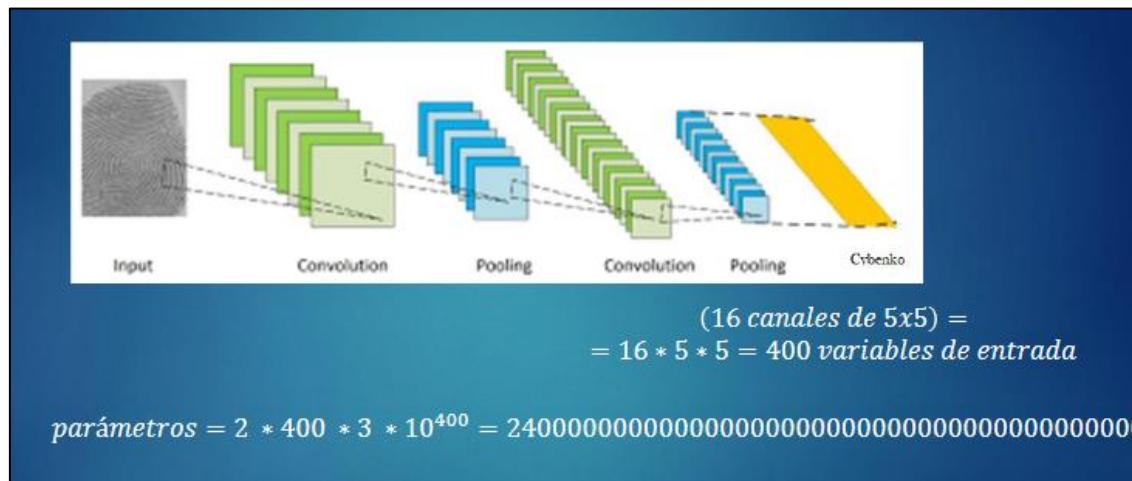
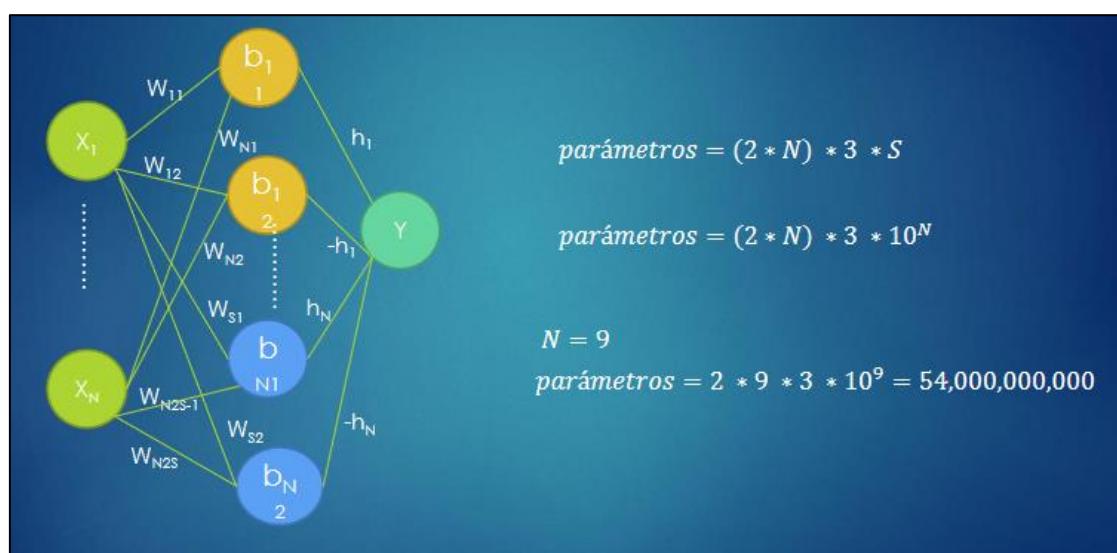
## Dos variables de entrada, un selector de región



## Dos variables de entrada ¿cuántos selectores de región?



## N variables de entrada S selectores de región

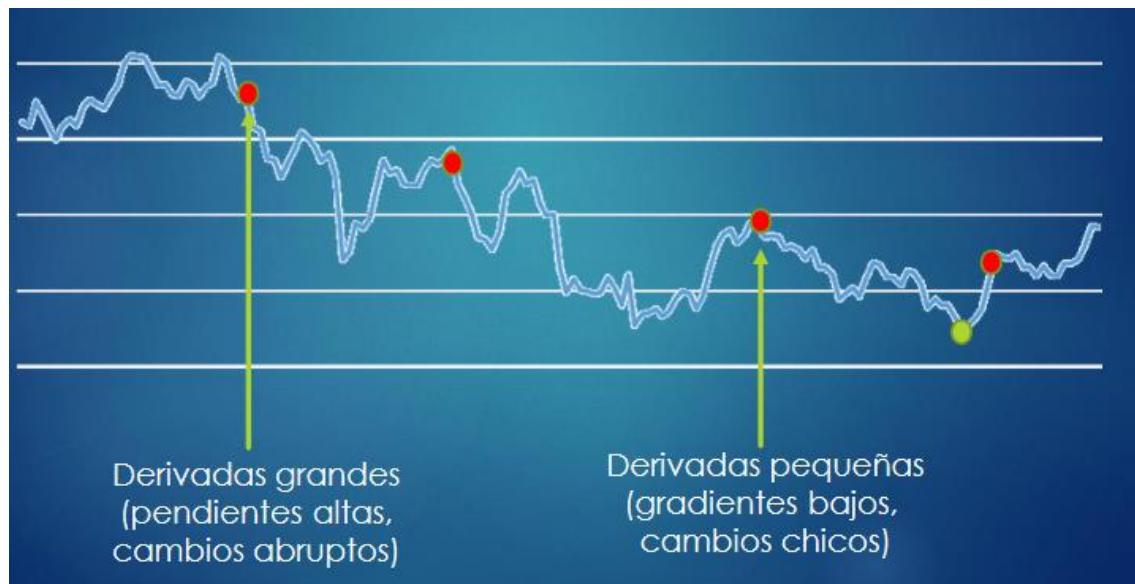


Entonces: la red ideal existe, pero no la vas a encontrar nunca por la cantidad de parámetros.

## Inicialización de pesos

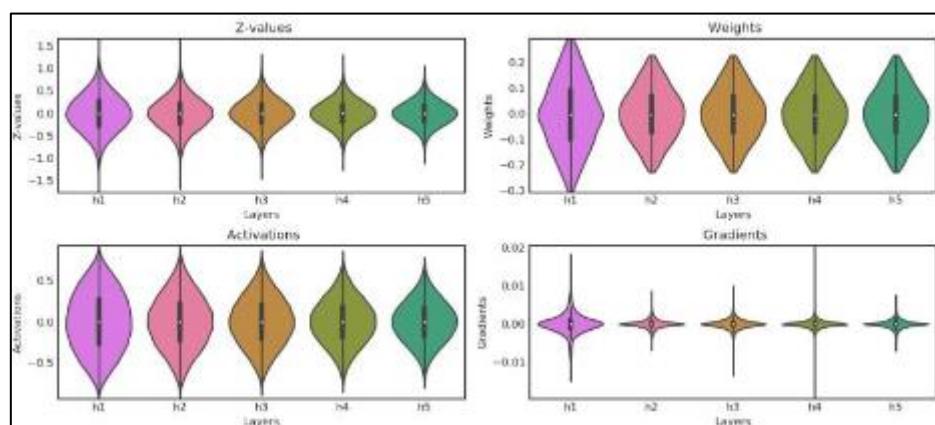
Una capa, superficie de error convexa.

Varias capas, superficie de error no convexa



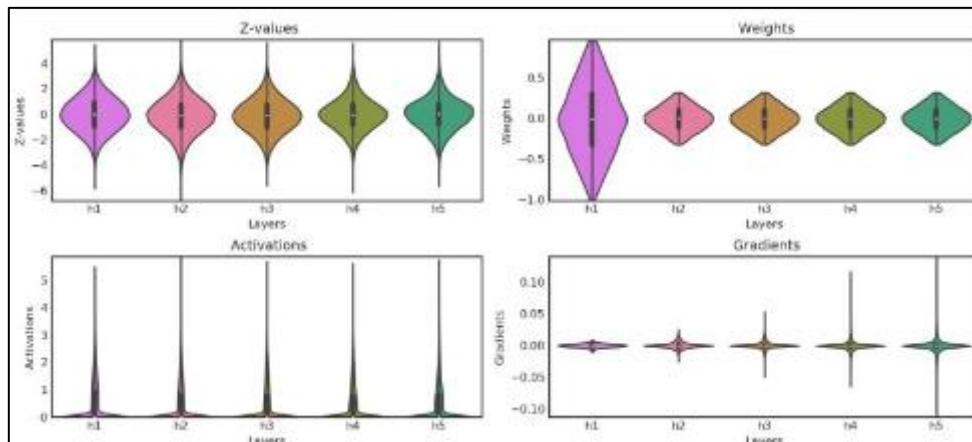
## Inicializador Glorot

- Funciona bien para capas con función de activación sigmoidea
- Magnitudes de los valores intermedios estables
  - o No tienden a 0
  - o Ni a valores muy grandes
  - o Derivadas estables
    - Evita el efecto del gradiente desvaneciente y el efecto del gradiente que explota.
- Glorot: Normal con media = 0 y Varianza =  $2 / (\#inputs + \#outputs)$ 
  - o #inputs = neuronas capa anterior
  - o #outputs = neuronas capa siguiente



## Inicializador He

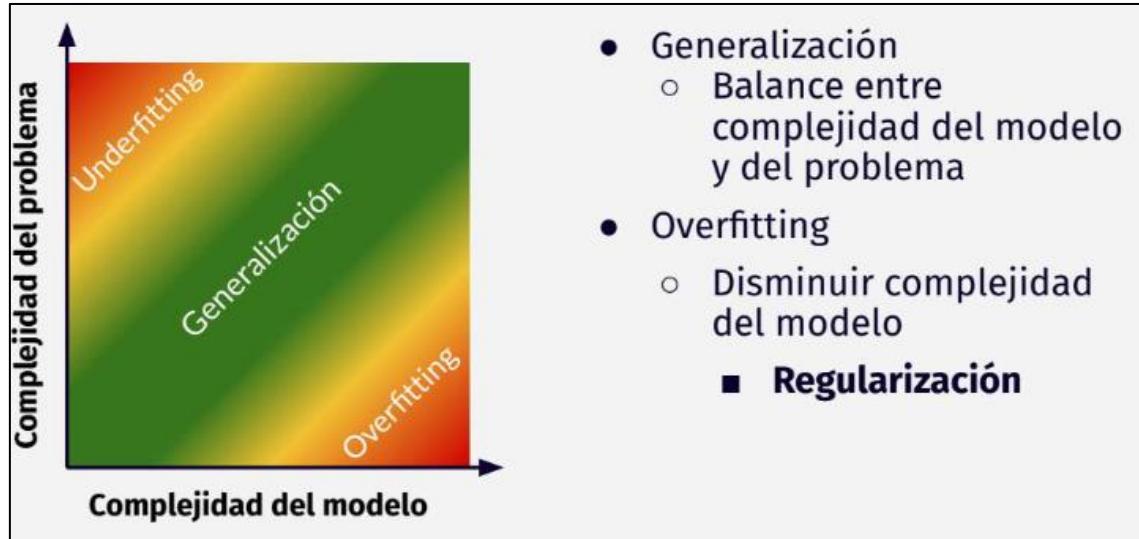
- Funciona bien para capas con función de activación ReLU
- Magnitudes de los valores intermedios estables
  - o No tienden a 0
  - o Ni a valores muy grandes
  - o Derivadas estables
- He: Normal con media = 0 y Varianza =  $2 / (\#inputs)$ 
  - o  $\#inputs$  = neuronas capa anterior



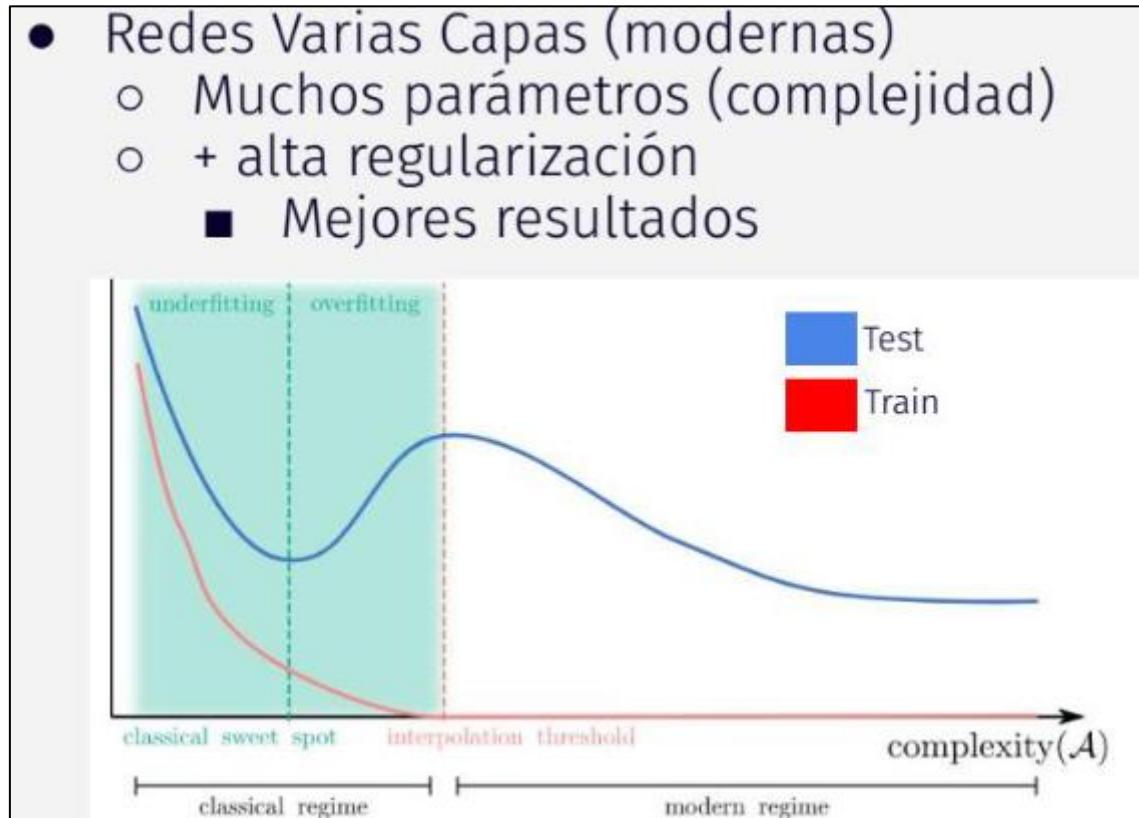
## Inicializadores en Keras

- random\_normal
- random\_uniform
- truncated\_normal
- zeros
- ones
- glorot\_normal
- glorot\_uniform
- he\_normal
- he\_uniform
- identity
- orthogonal
- constant
- variance\_scaling

## Regularizador de pesos



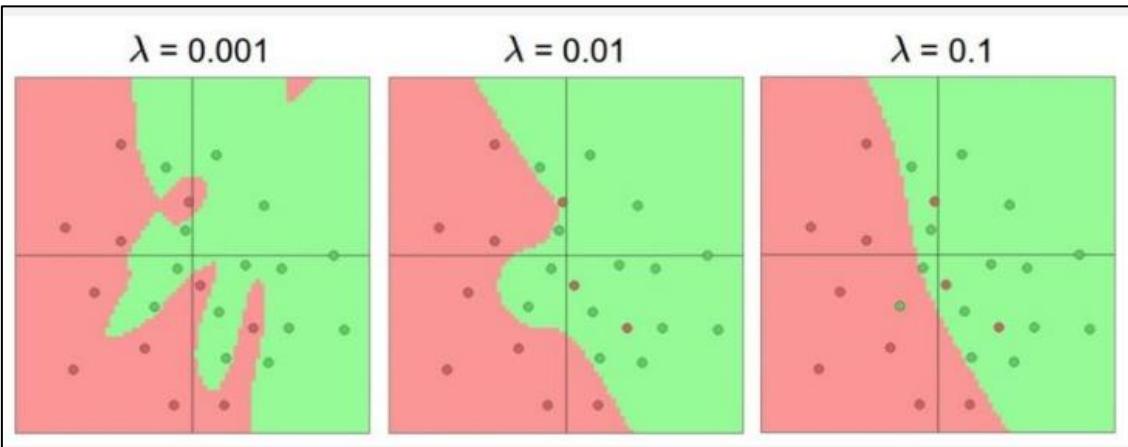
- Redes Varias Capas (modernas)
    - Muchos parámetros (complejidad)
    - + alta regularización
- Mejores resultados



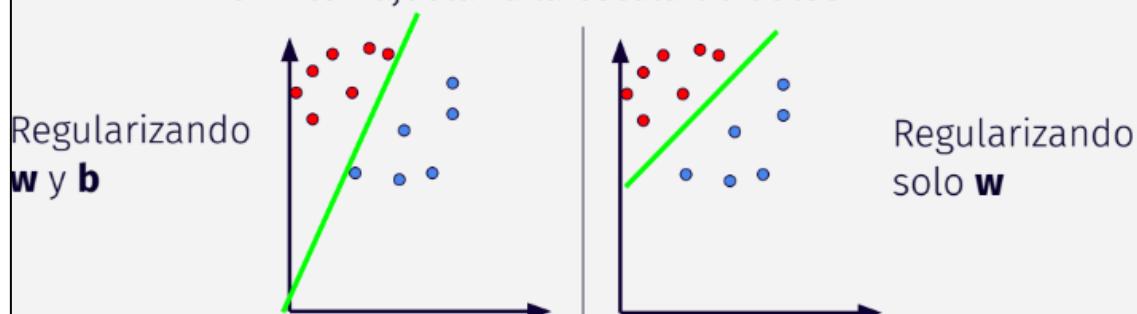
## Regularizador L2 (Norma euclídea)

- Pesos grandes (en magnitud)
  - Valores grandes
  - Predicciones extremas
    - Sobreajuste
- Función de error compuesta
  - Error de tarea +  $\lambda$  Error de regularización
    - $p_j^2$  penaliza a pesos grandes

$$E = \underbrace{\sum_i E_i}_{E \text{ de tarea}} + \lambda \underbrace{\sum_j p_j^2}_{E \text{ de regularización}}$$



- Pesos **w** de Dense o Conv
  - Si, **w** multiplica a entrada **x**
  - Genera valores extremos
- Sesgos **b** de Dense o Conv
  - No, sólo suman un valor a la salida
  - Permiten ajustar a la escala de datos



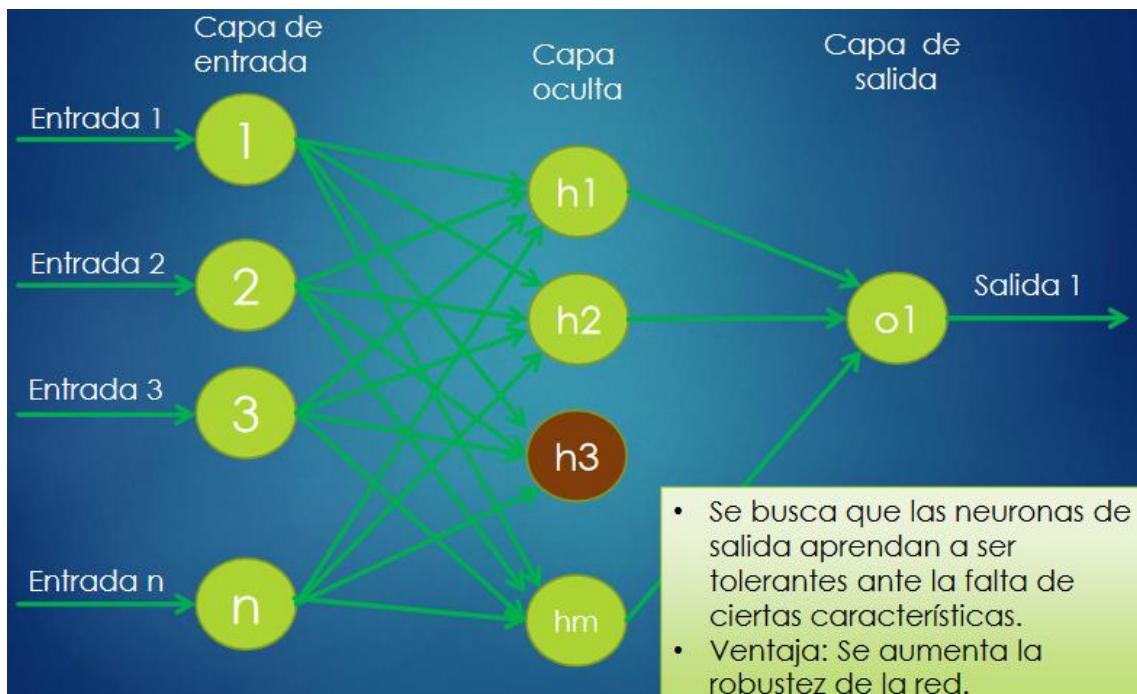
## Regularizador L1 (norma Manhattan)

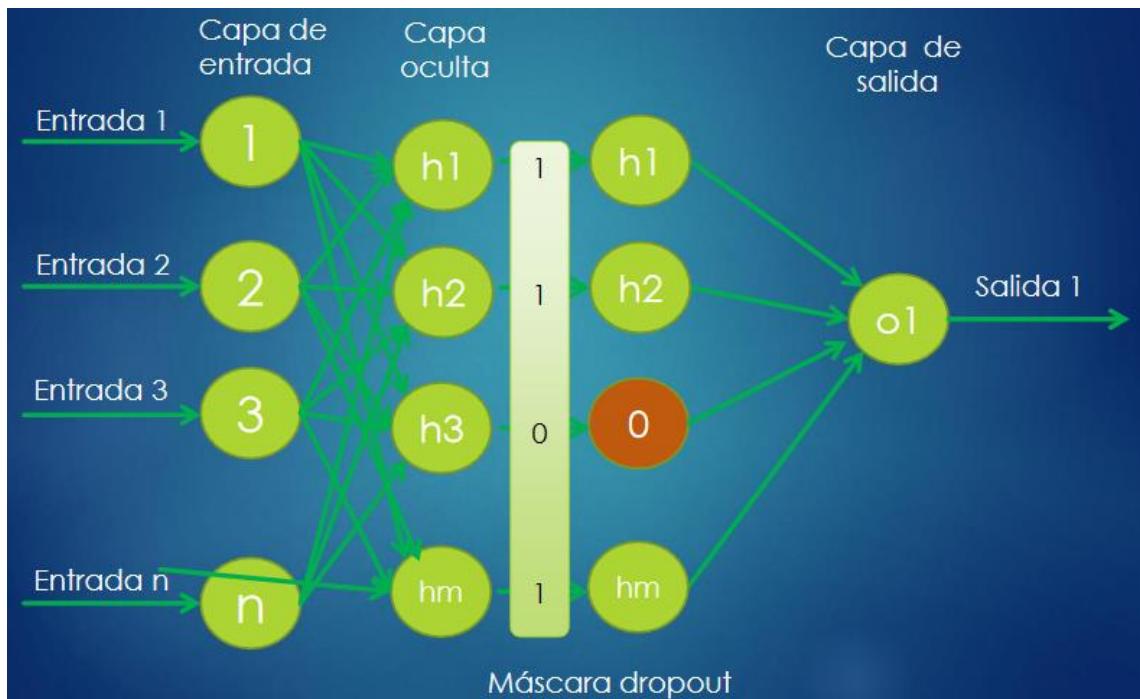
- $p_j^2$  puede ser muy fuerte
  - Utilizar valor absoluto  $|p_j|$
  - No es derivable en  $p_j=0$ 
    - Pero funciona

$$E = \sum_i E_i + \lambda \sum_j |p_j|$$

$$\begin{aligned}\delta E / \delta p_k &= \sum_i \delta E_i / \delta p_k + \lambda \sum_j \delta |p_j| / \delta p_k \\ &= \sum_i \delta E_i / \delta p_k + \lambda (\pm 1)\end{aligned}$$

## Capa dropout





- Las neuronas a activar cambian en cada batch
  - o La máscara de Dropout se recalcula
    - Cada neurona se activa con una probabilidad  $p$
    - $0 < p < 1$  es hiperparámetro de la capa
- Al entrenar:
  - o Las neuronas de salida están “acostumbradas a tener  $N \cdot p \leq N$  neuronas en promedio activadas”
- Al predecir:
  - o Las activations de entrada se multiplican por  $p$ 
    - Mantiene constante la magnitud promedio de los valores de entrada



## Ventajas

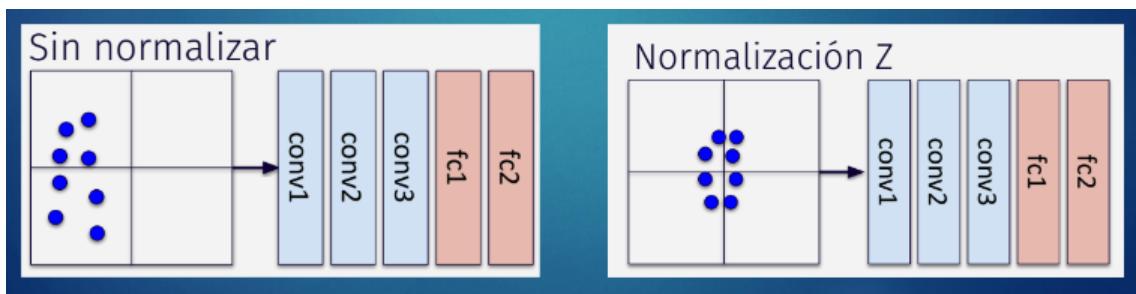
- Poco coste computacional
- Regulariza el modelo:
  - o Activaciones aleatorias

- Modelo distinto en cada batch
- Anular ciertas salidas efectivamente cambia la arquitectura.
- Valores de p: más bajo en capas más profundas

## Capa Batch normalization

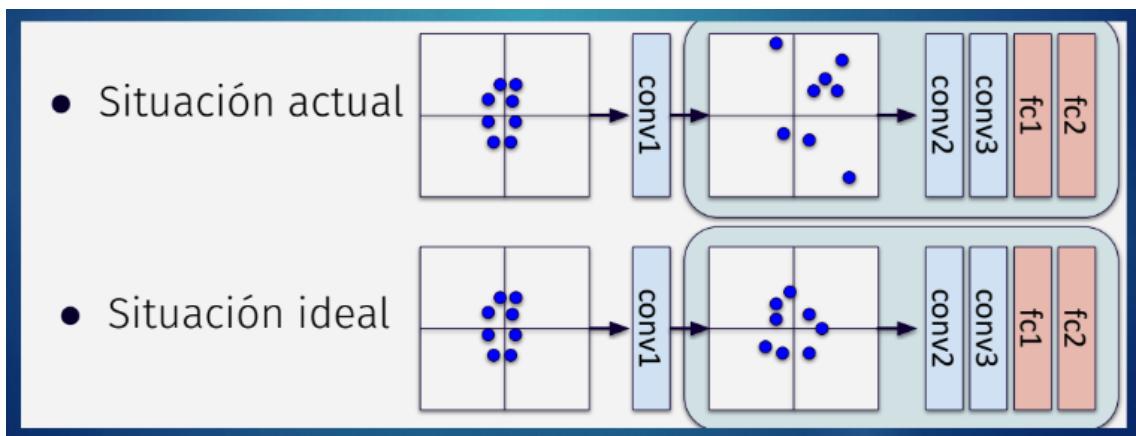
¿Por qué normalizamos los datos?

- Mejora el aprendizaje
- Mejora el accuracy
- Estandariza las magnitudes



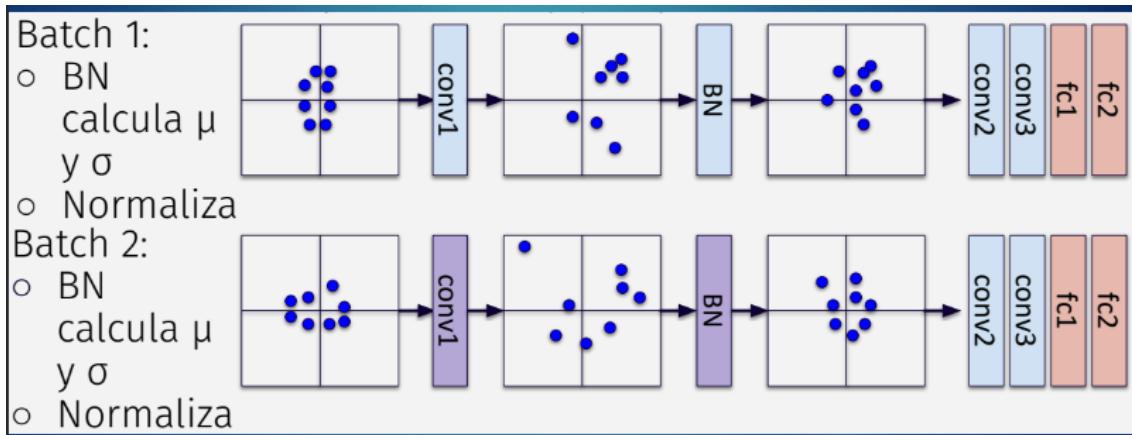
¿Qué pasa entre capas (conv2)?

- Una red sin su primer apéndice también es una red
- Normalicemos también la entrada a conv2



Normalización z en cada batch

- Estimar parámetros ( $\mu$  y  $\sigma$ ) para cada batch



## Balance de clases

En algunos problemas, se desea darle mayor importancia a minimizar los errores de algunos ejemplos, en particular las muestras de alguna clase “crítica”, por ejemplo en muestras positivas de alguna enfermedad.

En un problema de 2 clases (Clases A y B), por ejemplo, el error puede escribirse como:

$$E = C_A E_A + C_B E_B$$

donde  $E_A$  y  $E_B$  son los errores promedio de cada clase, y  $C_A$  y  $C_B$  son los pesos de cada clase. En general, los  $C_i$  se eligen de forma que sumen a 1.

Entonces, si se desea darle el doble de importancia a los errores de la clase A, se pueden elegir  $cA=0.66$  y  $cB=0.33$ , y el error global puede escribirse como:

$$E = 0.66 E_A + 0.33 E_B$$

En Keras los pesos de las clases se establecen al momento de llevar a cabo el entrenamiento.

## Optimizadores avanzados

### AdaGrad

- Normaliza la magnitud de los gradientes
  - Importa más la dirección
- Mejora el comportamiento con gradientes muy bajos/altos

### ADAM

- AdaGrad + Momentum
- La velocidad de aprendizaje no es tan importante

## Callbacks

- Uno de los principales problemas al entrenar redes neuronales, es elegir el número de épocas del entrenamiento.

- Si son excesivas la red podría tener overfitting
- Otro problema es cuando la red cae en un mínimo local de manera “temprana”.
- Keras permite **monitorear el entrenamiento** de una red, mediante el llamado a funciones “callbacks”.

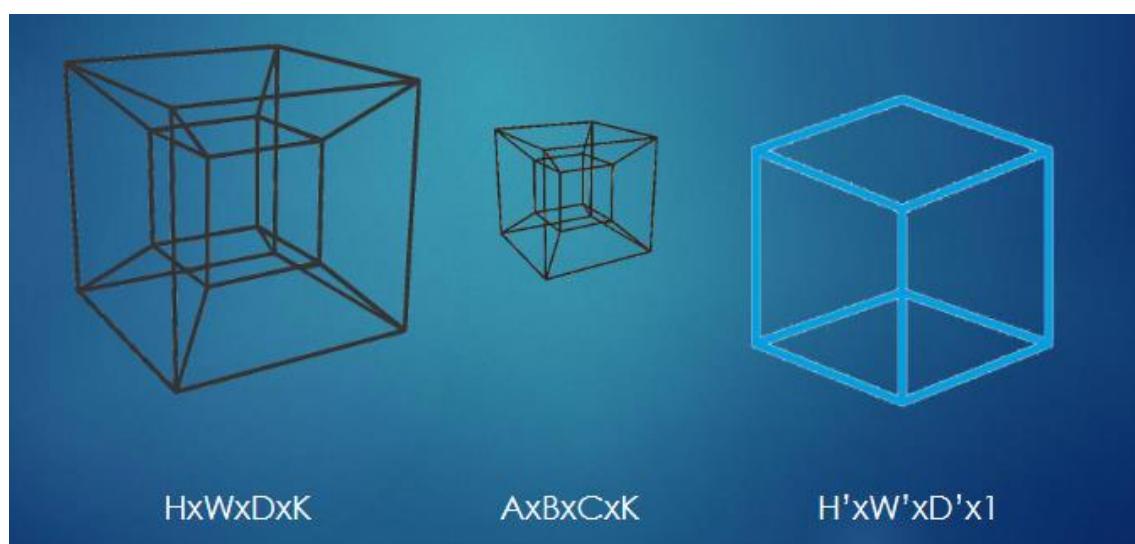
Los **callbacks** se invocan al principio o final de cada batch, época, testeo y entrenamiento.

## Earlystopping

Keras posee un Callback especial llamado EarlyStopping que **permite detener la ejecución** del entrenamiento de una red neuronal cuando una determinada métrica no cambia significativamente.

## Conv3D

- Los filtros se mueven en tres dimensiones.
- Se pueden usar para datos tensoriales (videos)
- La entrada puede tener canales (cuarta dimensión)

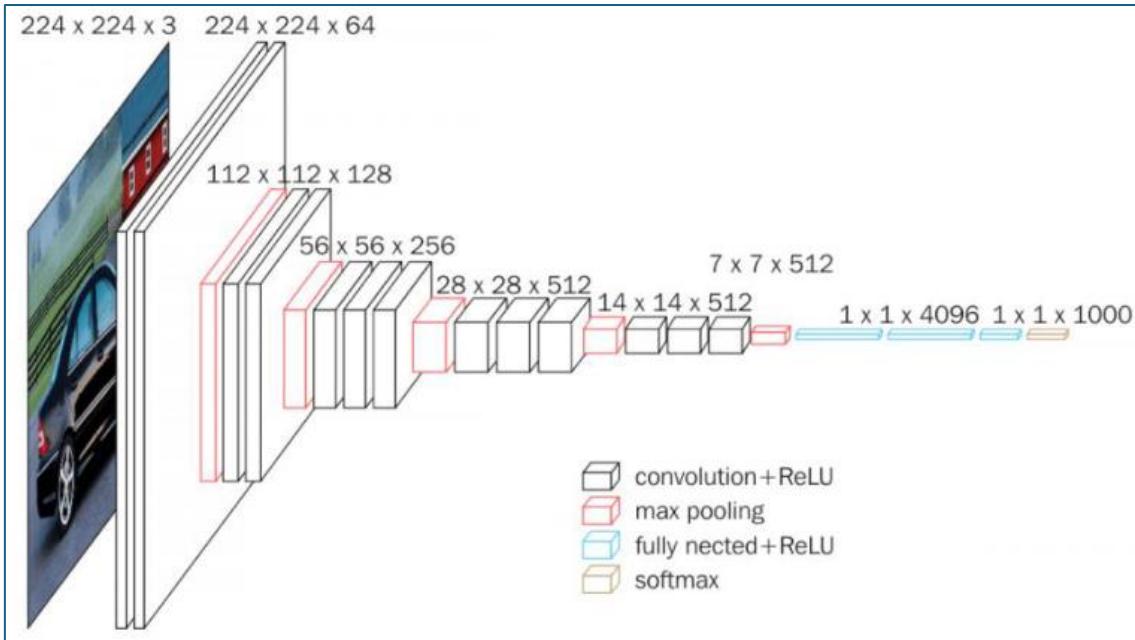


## Arquitecturas conocidas

### VGG (Visual Geometry Group, Oxford)

- Ganador de ILSVRC 2012 (competición ImageNet)
- Ideas principales
  - Muchas convoluciones 3x3
  - Diseño en bloques (varias Conv2D seguidas de un Pooling)
- 6 versiones
  - VGG D (16 capas) más popular (También llamada VGG16)
- 33M de parámetros
- Se utiliza mucho como parte de otros modelos
- Diseño en bloques: nueva forma de pensar las redes (preguntar)

- BGG16 y VGG19 disponible en Keras



## All convolutional

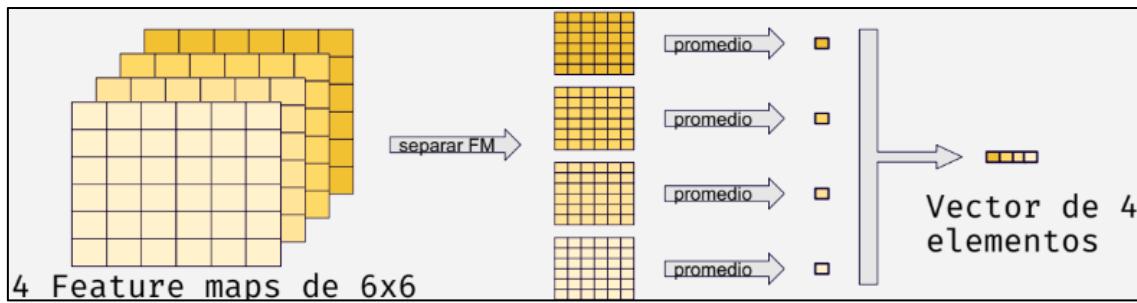
- Idea principal: solo convoluciones
  - o Sin MaxPooling => convoluciones con stride=2 ni capas Dense
- Probó que era posible usar sólo Convoluciones
- 3 versiones: La C es la más popular
- Capa especial: GlobalAveragePooling
  - o Reemplaza las Dense
- Más pequeña que VGG
  - o 1.3M de parámetros
- Diseñada para CIFAR10
  - o Más fácil que ImageNet
- Fácil de entrenar
- Buena performance
- Reducción de dimensionalidad espacial (32->16, 16->8)

## GlobalAveragePooling

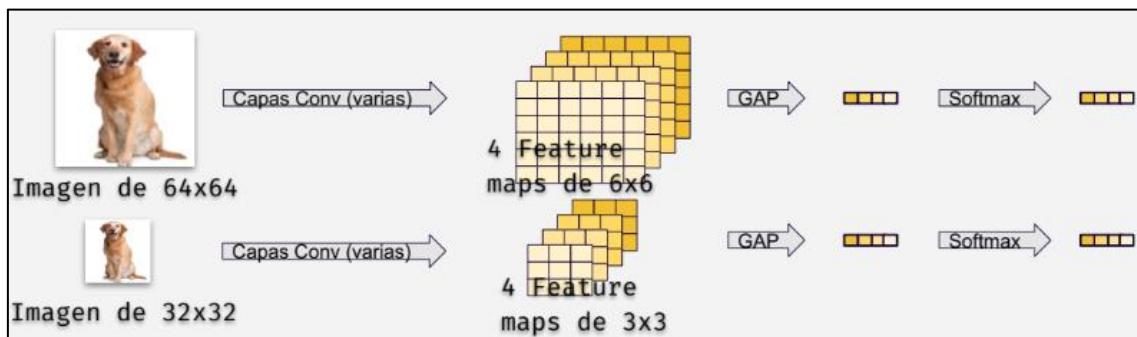
### Motivación

- Los feature maps se achican espacialmente
  - o Pero crecen en cantidad (#canales)
  - o Cada canal detecta una característica
- La dimensión espacial del feature map deja de importar

Método: promediar dimensiones espaciales



- Sucesivas capas con stride > 1 achican HxW a H'xW'
- La capa GlobalAveragePooling borra las dims H'xW'
- No importa la resolución original HxW



- Permite calcular puntajes para cada clase
- Reemplaza las Dense/Flatten
- Promedia las dimensiones espaciales
  - o Deja solo la dimensión de canales de un feature map
- Primero se suele utilizar una conv 1x1
- Tantos feature maps como clases
- Ejemplo si tengo 10 clases:
  - o Convolución (1x1) con 10 feature maps de HxW
  - o HxWx10 → la salida de GAP es de 10 elementos

## MobileNet

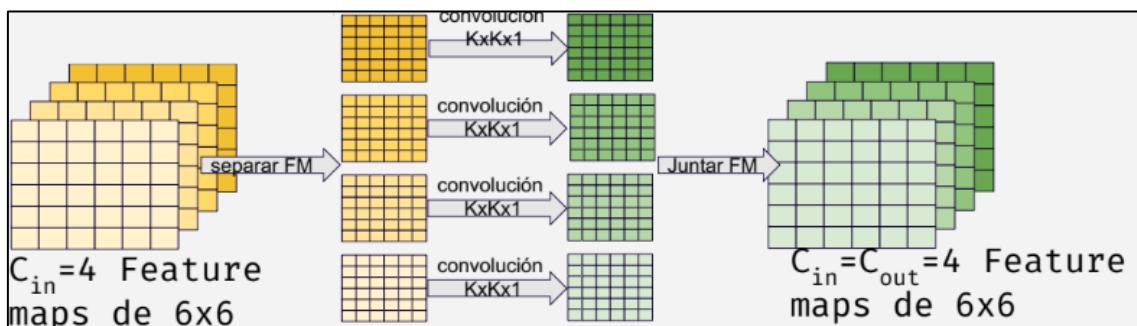
- Eficiencia en mente
  - o Convoluciones “Separables” (Depthwise)
- Poco uso de memoria y CPU
  - o Dispositivos móviles
- Relación cantidad feature maps / tamaño similar a VGG
  - o  $112 \times 112 \times 32 \Rightarrow 56 \times 56 \times 128 \Rightarrow 28 \times 28 \times 256 \Rightarrow 14 \times 14 \times 512 \Rightarrow 7 \times 7 \times 1024$
- GlobalAveragePooling para quitar dimensiones espaciales
  
- 4 bloques con features maps de 64, 128, 256, 512
- Cada bloque:
  - o SeparableConv2D(n,(3,3), stride=(1,1))
  - o SeparableConv2D(n,(3,3), stride=(2,2))
- SeparableConv2D(1024,(3,3)) al final
- AvgPooling

- Pero le agregan una capa Dense (FC)
- Diseñada para ImageNet
  - Imágenes de 224x224x3
- Disponible en Keras

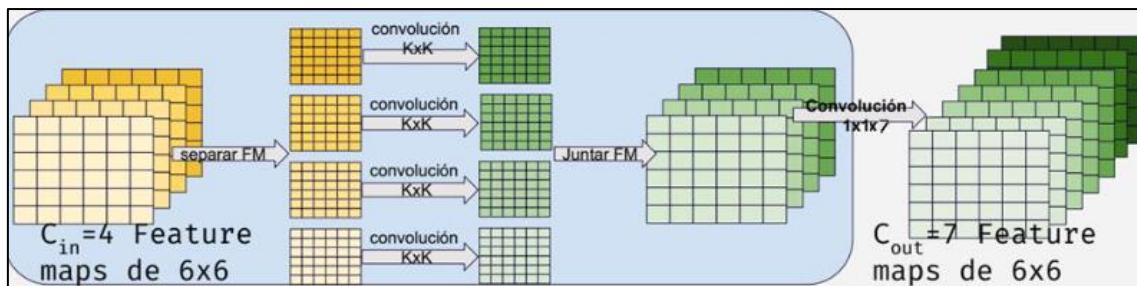
## Capa de convolución separable en profundidad

Toman un feature map de 4 canales. En un kernel normal eso se transforma en un solo canal a la salida. El kernel depthwise te retorna la misma cantidad de canales, pues realiza una convolución en cada canal por separado.

- Convoluciones comunes: transforman  $H \times W \times C_{in}$  en  $H' \times W' \times C_{out}$ 
  - $C_{out}$  filtros de tamaño  $K \times K \times C_{in}$ 
    - Generan  $C_{out}$  canales de salida
    - “Mira” los  $C_{in}$  canales de entrada
- Convolución Separable: Cada filtro mira un solo canal de entrada → Filtros  $K \times K \times 1$



- Convoluciones separables: transforman  $H \times W \times C_{in}$  en  $H' \times W' \times C_{in}$
- Agregar una convolución  $1 \times 1$  (depthwise) para generar  $H \times W \times C_{out}$
- Menos parámetros
- Menos cómputo
- Menos poder de representación (no mucho)



MobileNet y MobileNetV2: disponible en Keras

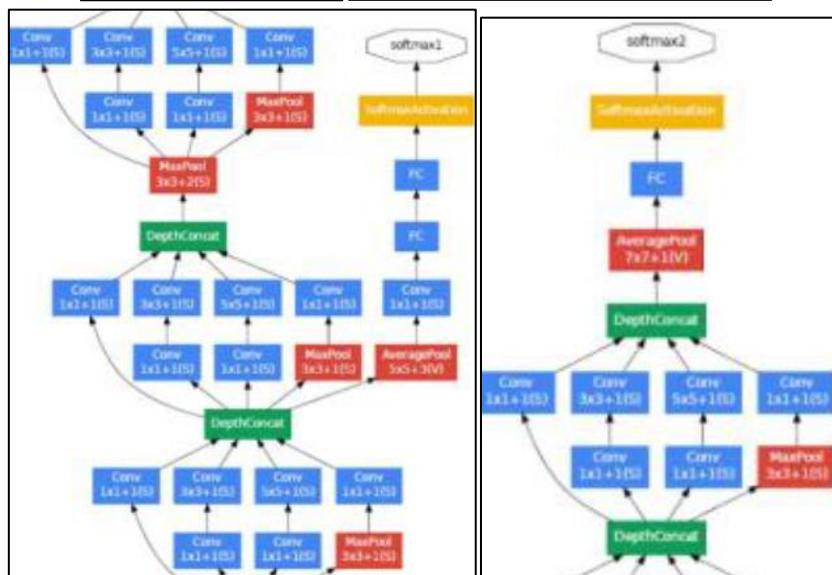
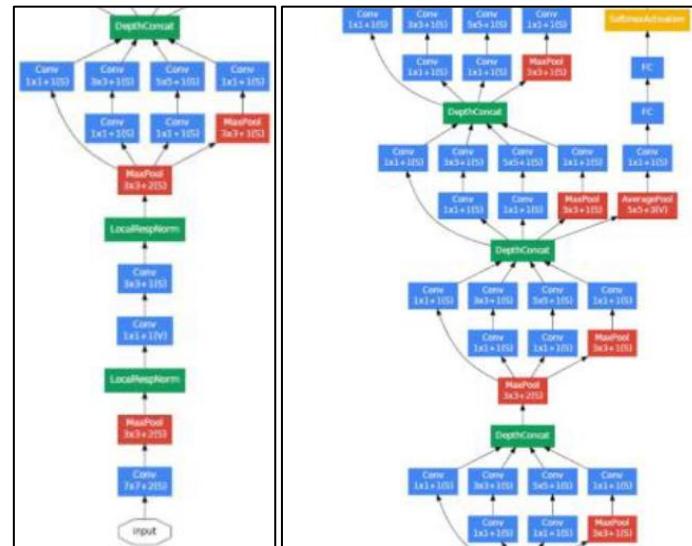
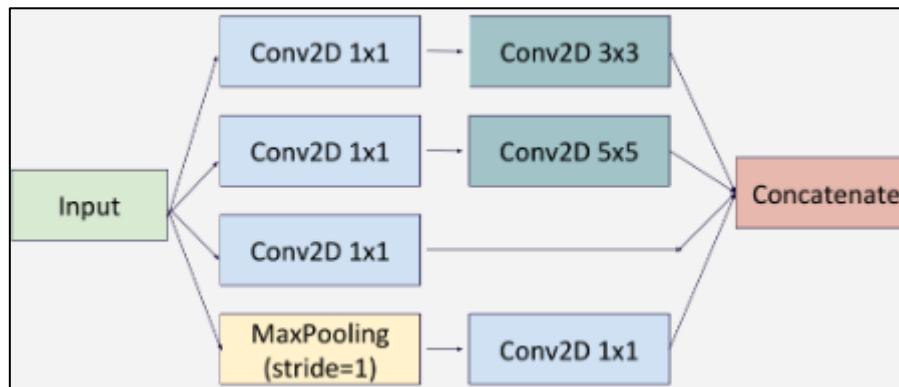
## Inception

- Ganador de ILSVRC 2014 (ImageNet)
- También conocido como GoogleNet
- Introdujo bloques Inception

## Bloque inception

- Hasta ahora se buscaba mayor profundidad
- Otra forma → Mayor ancho

- Mayor diversidad
- Bloques Inception
  - Conv + eficientes
  - Distintas convoluciones
  - Mismo tamaño HxW
  - Concatenar dimensión C



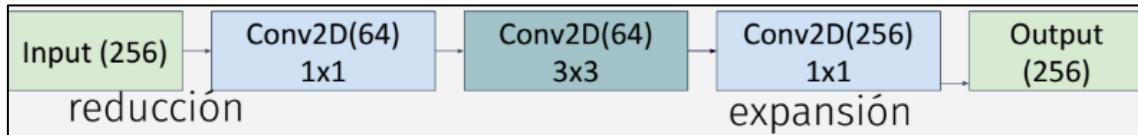
## Residual Networks (ResNets)

Primeras en entrenar redes con hasta 152 capas

### Bloques Bottleneck

Objetivo: menor coste computacional

- $1 \times 1 \times 256 + 3 \times 3 \times 64 + 1 \times 1 \times 256 = 1088$  parámetros vs.
- $3 \times 3 \times 256 = 2304$

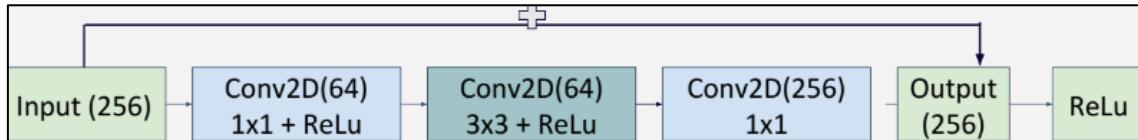


### Bloque residual

- Aprenden una modificación de la entrada
- Al comenzar el entrenamiento, la capa es la función identidad
- La salida es igual a la entrada (mismas dimensiones)
  - o Se pueden usar muchas capas más
- En el peor caso, no hacen ninguna modificación



### Bloques bottleneck+residuales



ResNet50, ResNet101 y ResNet152 disponible en Keras

## Transferencia de aprendizaje

Entrenar un modelo desde cero requiere:

- Tiempo/uso de cpu
- Optimización de hiperparámetros
- Preparación de datos

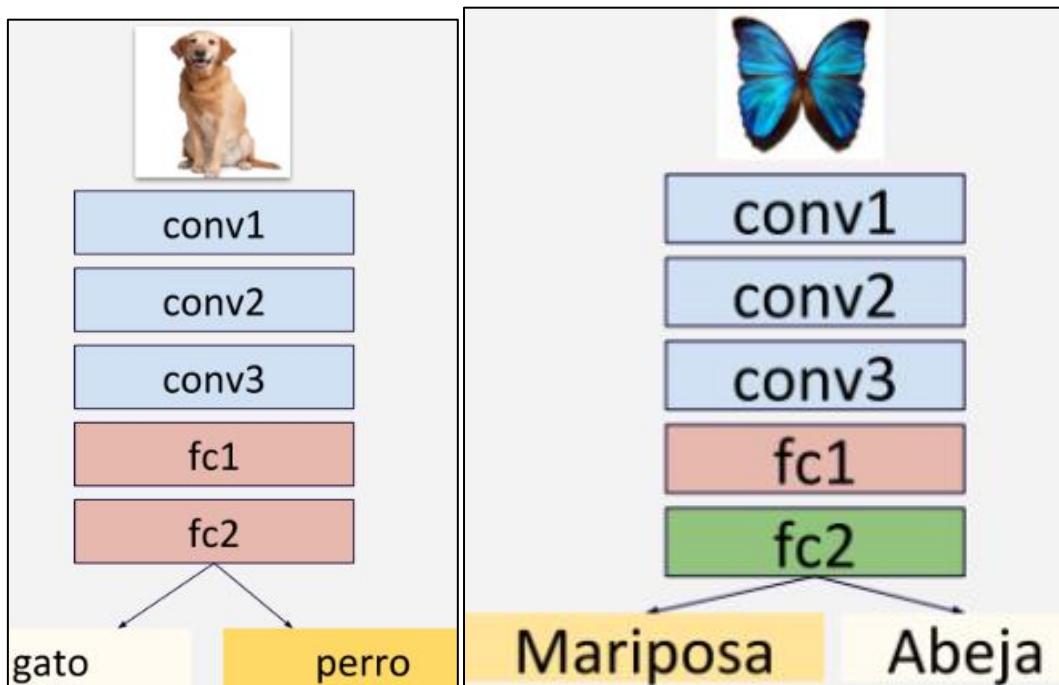
Al usar un modelo preentrenado

- Resulta más fácil y rápido
- Arquitectura fija
- Dominio fijo

Tengo un modelo que resuelve bien el problema X y ahora quiero resolver un problema Y que es muy parecido (mismo dominio)

Ejemplo:

- Tengo un modelo que clasifica gatos y perros
- Ahora quiero un modelo que clasifique mariposas y abejas



- Reusar filtros convolucionales
- Relativamente independientes del dominio (las primeras capas)
- Reemplazar últimas capas
- Reentrenar red
  - o “Congelar” capas anteriores (las capas no se entrena)

## Finetuning

Reentrenar todo el modelo

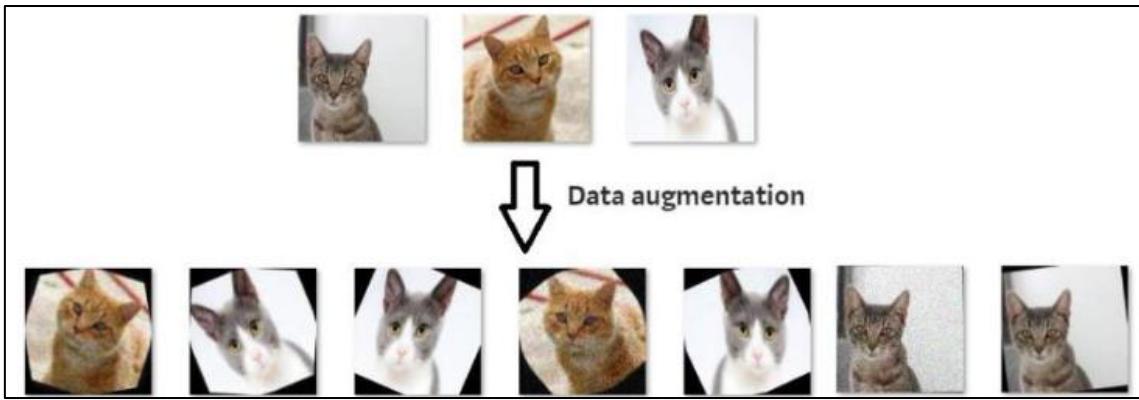
- Más tiempo de entrenamiento
- Resultados ligeramente mejores (1% a 5%)

Código: igual al ejemplo anterior pero sin congelar pesos

## Aumentación de datos

Generar nuevos ejemplos

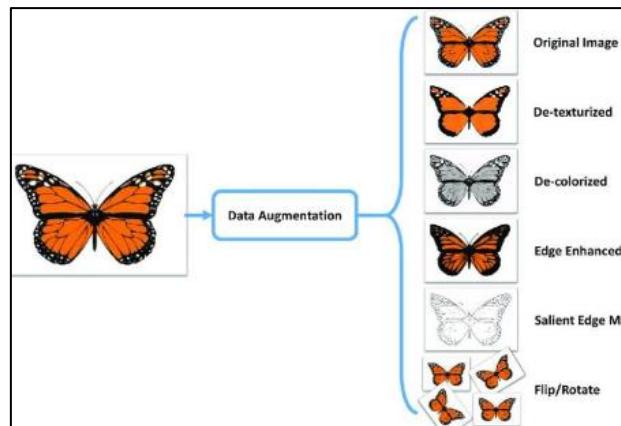
- Definir transformaciones útiles
- Transformar los ejemplos disponibles



- Rotaciones
- Desplazamientos
- Voltear
- Recortes
- Oclusiones
- Cambio de fondo



- Cambios de brillo, color y textura



## Interpretabilidad y explicabilidad

**¿Qué es la interpretabilidad?**

Es un campo de estudio de la IA, cuyo objetivo es estudiar técnicas para comprender los estímulos por los cuales los modelos de caja negra generan determinadas salidas.

Utilizada para dar un rol diagnóstico en descubrir cómo es la contribución las capas ocultas en los modelos, y así brindando una forma de entenderlos.

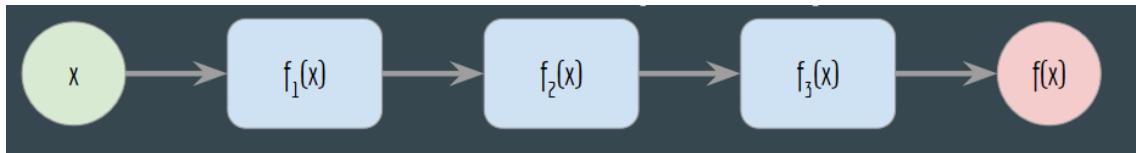
La interpretabilidad pasa a ser un requisito, especialmente para sistemas que toman decisiones de alto riesgo → proveer de feedback a los usuarios finales respecto a cómo se generaron las salidas de los modelos.

#### Tipos de interpretabilidad

- Post-hoc
- Intrínseca

### Estado del arte de métodos de interpretabilidad

#### Vista relevante de una red neuronal para interpretabilidad

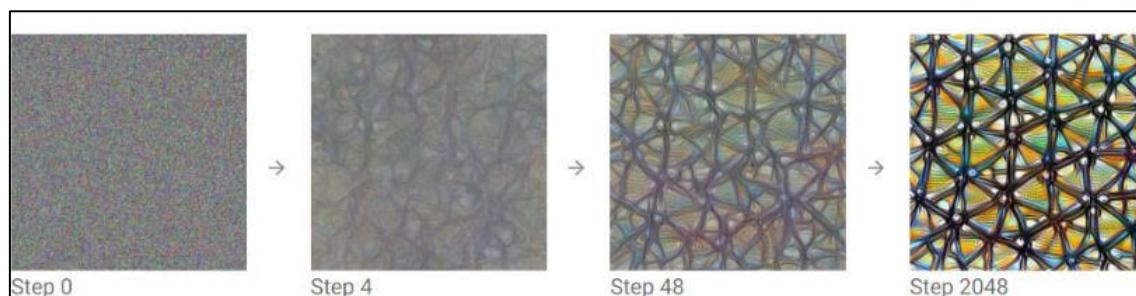


- Valores intermedios o features/activaciones
- Posibles derivadas/gradienes
  - o Cómo afecta la entrada a la salida
  - o Cómo afecta la entrada al feature i
  - o Cómo afecta el feature i a la salida
  - o Cómo afecta al feature i al feature j

### Algunos enfoques para interpretar modelos

#### Generativos

- Permiten visualizar y comprender las características que aprenden los modelos
- Su funcionamiento se basa típicamente en un ascenso de gradiente
  - o Maximizamos una función (capa) interna de la red → aprendemos lo que le “gusta” recibir a la red
  - o Esto se logra a partir de generar ejemplos que lleven al modelo a clasificar la clase objetivo
- Encontrar entrada con mejor puntaje: argmax score(x) [ej. softmax de la salida]
  - o Generar vector de ruido x
- Iterar hasta converger



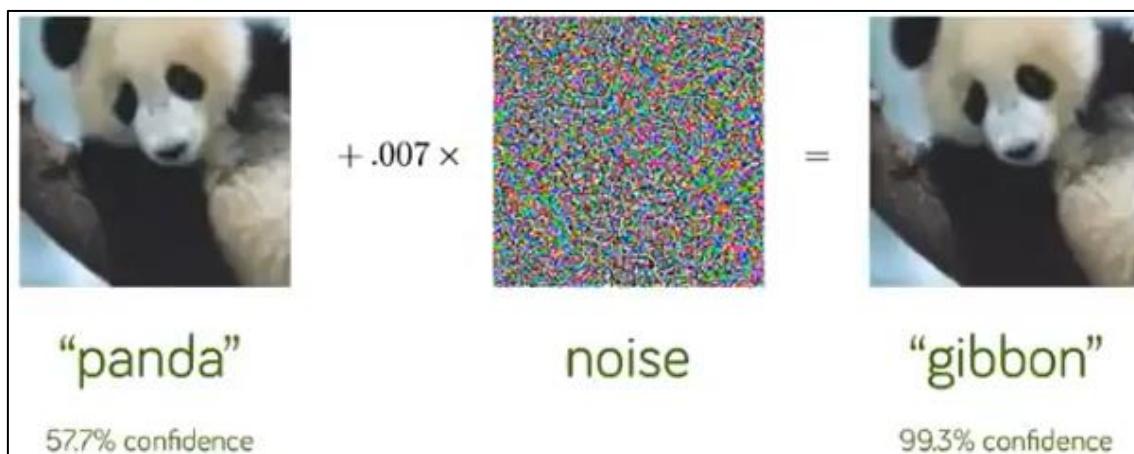
- Impacto de la función de puntaje (dónde se aplica el ascenso de gradiente)
  - o Primary Attribution: Evalúa la contribución de cada feature a la salida de un modelo.
  - o Layer Attribution: Evalúa la contribución de cada neurona en una capa dada a la salida del modelo.
  - o Neuron Attribution: Evalúa la contribución de cada feature en la activación de una neurona oculta en particular.
- La optimización nos permite responder a preguntas como
  - o ¿Qué está buscando la red? ¿En qué se centra X parte de la red?

## Métodos por activación

- Simplemente visualizar los features  $f(x)$
- Problema: demasiados features
  - o Se pueden ordenar por magnitud

## Métodos adversarios

- Permiten validar la robustez y vulnerabilidad del modelo
- Se perturba la imagen de entrada intencionalmente para engañar al modelo a clasificarla como otra clase (objetivo)
- Método muy usado: Projected Gradient Descent (PGD)
  - o Elegimos ejemplo  $X$  de clase  $C$ , elegimos clase  $K \neq C$  objetivo
    - Iterar:  $x = x + \alpha \frac{\delta p_{\text{je\_clase}}(x, K)}{\delta x}$  hasta que  $p_{\text{je\_clase}}(K) > p_{\text{je\_clase}}(C)$
  - o Resultado: obtenemos la tendencia general de preferencia del modelo.



## Aproximaciones Post-hoc

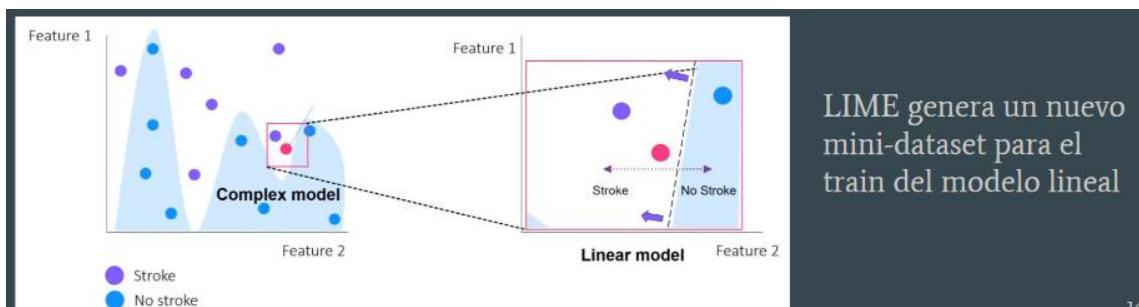
### (pixel) feature attribution approaches

- Resaltan los píxeles que fueron relevantes para la clasificación de una imagen en cuestión
  - o Explican predicciones (probabilidades) individuales, atribuyendo a cada feature de entrada (píxel) cuándo influyó en el cambio de la predicción de una clase (negativa o positivamente)
- Basados en derivadas (o gradientes)
- Basados en perturbaciones a la entrada (ej. occlusiones)

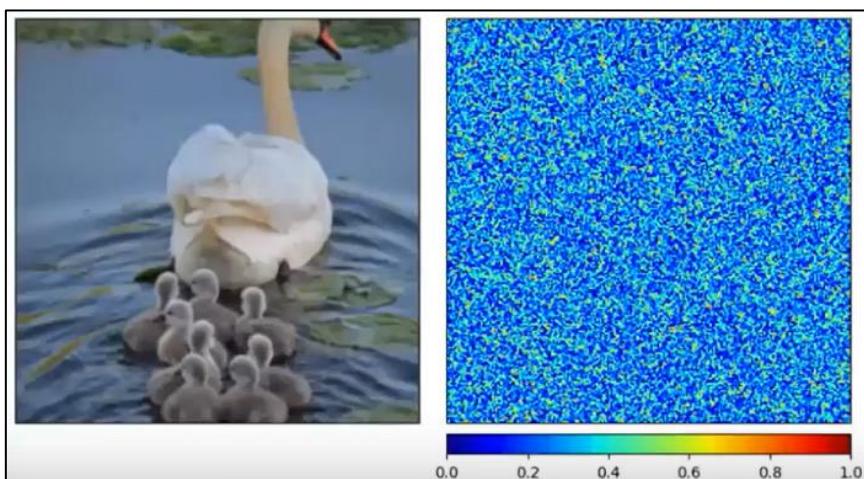
- Si un método es para modelos black-box, implica que no se requieren acceder a sus parámetros internos (pesos)

## LIME (Local Interpretable Model-agnostic Explanation)

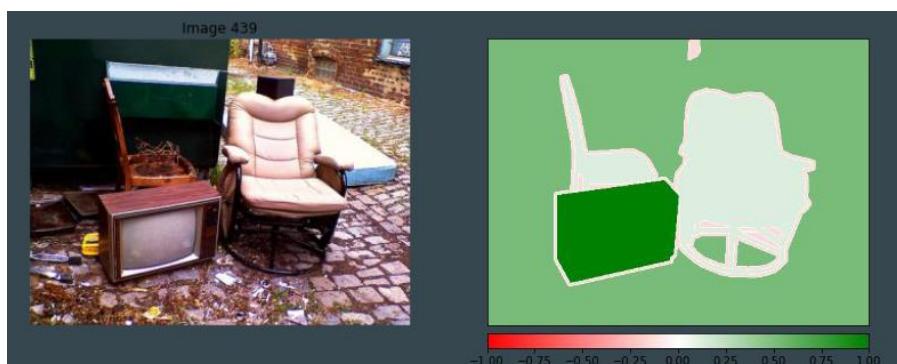
- Black-box, post-hoc, local, model-agnostic
- Utiliza la técnica de “sustitución” (surrogate) de modelos para obtener predicciones aproximadas localmente, las cuales son explicables
- Se entrena un modelo sustituto “lineal y explicable” (ej. regresión lineal) para una zona local (vecindario) para una predicción individual y específica



- Para imágenes no funciona, si con datos tabulares.



- Se puede usar el parámetro feature\_mask (dataset w/segmentations) (chequear)



## SHAP (Shapley Additive exPlanations)

Surge de la Teoría de Juegos Cooperativos => Valores de Shapley

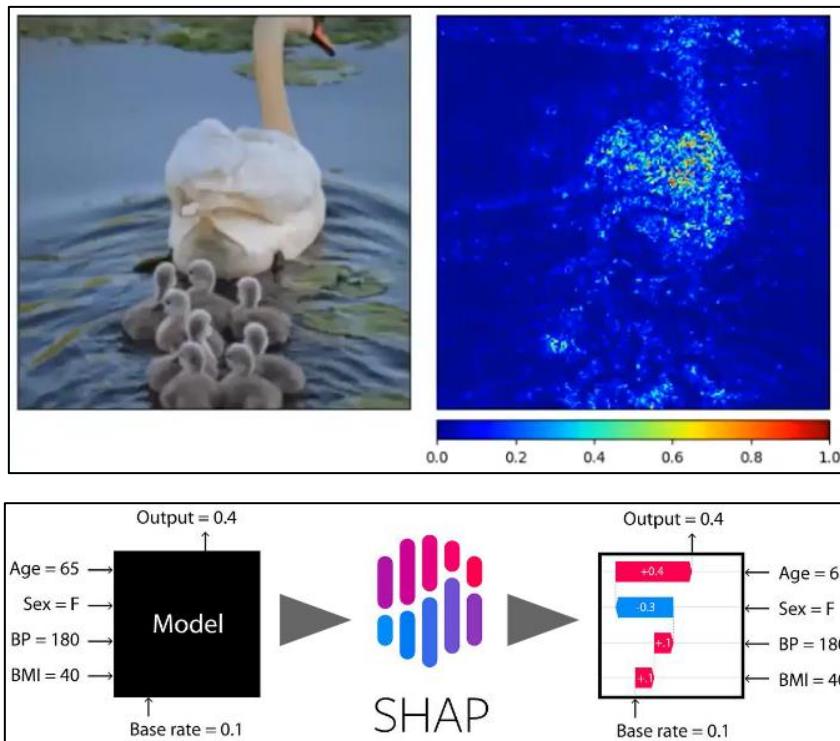
- Ejemplo: hay un grupo de personas que COOPERAN en un juego ganar un premio de 10k\$
  - o Buscamos una solución justa (fair)
- Estos valores nos indican el promedio de contribución de un jugador al equipo para el premio
  - o Llamado “Marginal value” (al grupo de jugadores se lo conoce como “coalition”)

Intuición: comparar el resultado con y sin un jugador específico, así ver cómo éste contribuyó al equipo para ganar

- ¡Alto ahí vaquero! Calcular la contribución individual no siempre sirve...
  - o Habría que considerar subconjuntos de jugadores, ya que Batman no es lo mismo sin Robin.
  - o Aunque puede suceder que la agrupación de un subconjunto no tenga relevancia, por ejemplo, en el subset (Messi, Dybala, Alvarez), el que más aporta ahí es Messi por sí solo (no requiere del resto para contribuir).

¿Cómo se realiza la inyección de este concepto al aprendizaje automático?

- Jugadores = Features
  - o Cada feature puede contribuir diferente al resultado
- Resultado = predicción del modelo
- El juego/competición = el modelo de caja negra



SHAP: black-box, post-hoc, local (podría ser global), model-agnostic

- Nos indica como estas predicciones son fairly-distributed entre las entradas individuales

- Kernel Shap: utiliza una función kernel para aproximar el valor de Shapley (similar a LIME)
  - o Más rápido que Gradient SHAP, pero menos preciso
  - o Kernel-trick/method: utilizar algo lineal para resolver algo no-lineal (ej. lo usan los SVM)
- Gradient Shap: calcula el valor de Shapley directamente añadiendo ruido gaussiano a las features de entrada y calculando después el gradiente de la salida con respecto al ruido.
  - o Más preciso que Kernel SHAP, pero más lento.

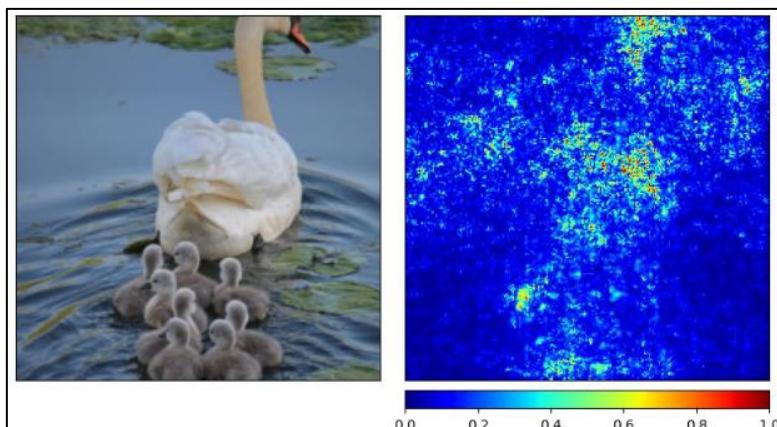
## Gradient-based approaches

### Saliency (Maps) o Vanilla Gradients

- Método que genera un mapa de importancia (saliency) mediante una sola iteración de backpropagation a la red.
  - o Black-box, post-hoc, local, model-agnostic
- Se basa en el cálculo del gradiente del puntaje de una clase con respecto a la imagen de entrada
- Si  $f(x)$  es un modelo de caja negra, x una imagen:
  - o  $\delta f(x)/\delta x$ : cómo cambia la salida (completa) del modelo al cambiar un pixel de la entrada
  - o  $\delta \text{Score}_c(x)/\delta x$ : cómo cambia la salida para la clase  $c$  del modelo ... idem
- La salida de la red es un vector de scores  $S(I)=[S_1(I), \dots, S_C(I)]$  (de largo  $C$ )  $C$ : # de clases (ej. 1000 en ImageNet)
- La entrada  $I \in \mathbb{R}^{n \times m}$  (imagen de  $n \times m$  píxeles)

$$E_{grad}(I_0) = \frac{\delta S_c}{\delta I} |_{I=I_0}$$

- La idea es similar a la de backpropagation (partial derivatives + chain rule)
  - o En lugar de calcular el gradiente de  $dE/dw$  (o sea, del loss), calculamos el gradiente del score (que sería el gradiente de la función de error para la clase particular  $c$  que nos interesa)
- Nota: al derivar con respecto a  $I$ ,  $\delta S_c / \delta I$  va a tener el mismo shape que  $I$



### Grad-CAM (Gradient-weighted Class Activation Mapping)

Similar a Saliency, pero el backpropagation no se hace hasta la imagen de entrada sino hasta la última (no necesariamente) capa convolucional

- GC se basa en la hipótesis de que como los feature maps de las capas de las CNNs aprenden patrones en los features de entrada, si se analiza las regiones activadas de los feature maps de la última capa se puede llegar a entender en qué se fija el modelo
  - o Igual siguen existiendo una enorme cantidad de features maps en la última capa (mismo problema que en los métodos de activación)
  - o ¿Cómo podríamos visualizarlo? Haciendo el promedio
    - Así reduciendo la dimensionalidad

**Problema 1:** los features maps de la última capa tienen información de TODAS las clases, y a nosotros solo nos interesa una clase particular para una imagen específica.

- GC pondrá los features maps con el gradiente del score para la clase de interés con respecto al feature map k

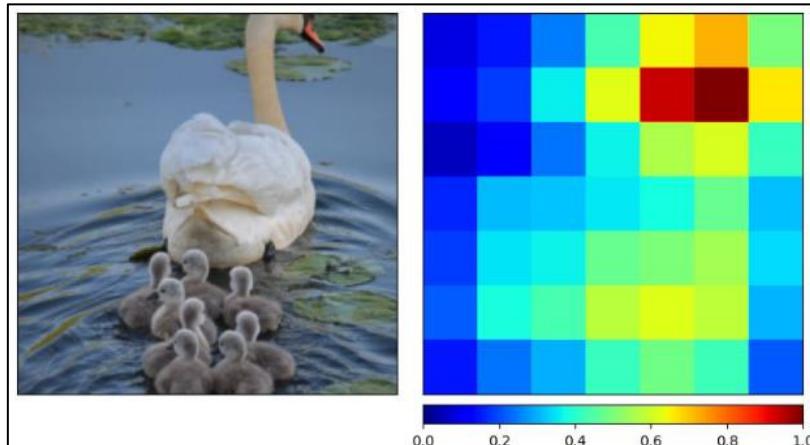
**Problema 2:** los feature maps son más chicos que la imagen de entrada... se puede resolver haciendo una interpolación (Guided Grad-CAM) y así conseguir su proyección a la imagen de entrada

- $\delta f(x)/\delta \text{feature\_map}_i(x)$ : cómo cambia la salida con respecto al feature map i
- Se usa una ReLU con el propósito de solo quedarnos con las partes del feature map que contribuyen a la clase c

$$L_{\text{Grad-CAM}}^c \in \mathbb{R}^{u \times v} = \underbrace{\text{ReLU}}_{\text{Pick positive values}} \left( \sum_k \alpha_k^c A^k \right)$$

$$\alpha_k^c = \text{average} \underbrace{\left( \frac{\delta S_c}{\delta A_{ij}^k} \right)}_{\text{gradients via backprop}}$$

Global Average Pooling



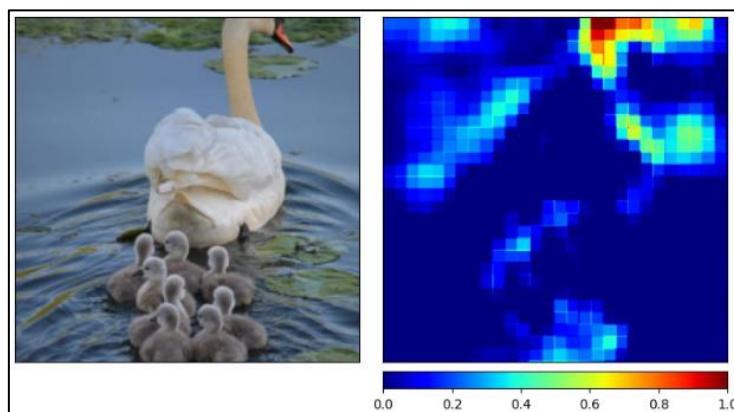
## Perturbation-based approaches

### Occlusion

- Black-box, post-hoc, local, model-agnostic
- Genera máscaras que ocultan solamente un parche de la imagen por vez.
  - o Se lo puede pensar como un tipo de ataque adversario
- El proceso es similar al operador de convolución, con un tamaño fijo de ventana y de salto

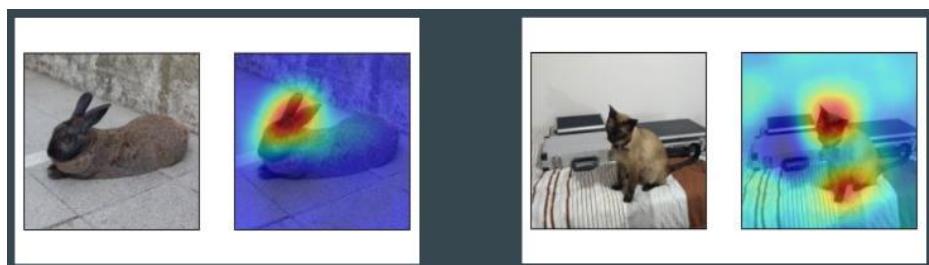
$$\sigma = ||f(I) - f(I_{masked})||$$

Si  $\sigma$  es grande, entonces el parche ha de ser importante ya que removérselo genera un gran cambio en la salida.



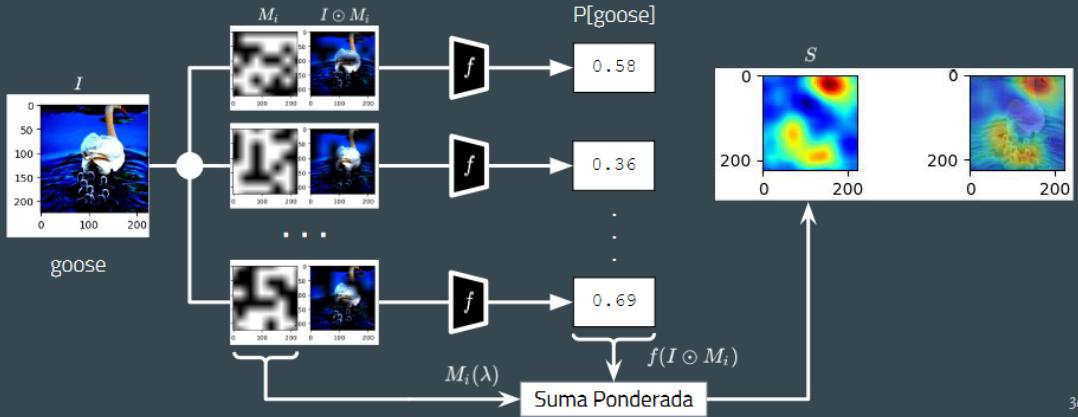
### RISE (Randomized Input Sampling for Explanation)

- Black-box, post-hoc, local, model-agnostic
- Es Occlusion + SHAP, pero suavizado
  - o Pro: mejora a Occlusion ya que oculta varios parches a la vez
  - o Contras: tarda mucho tiempo en computar...



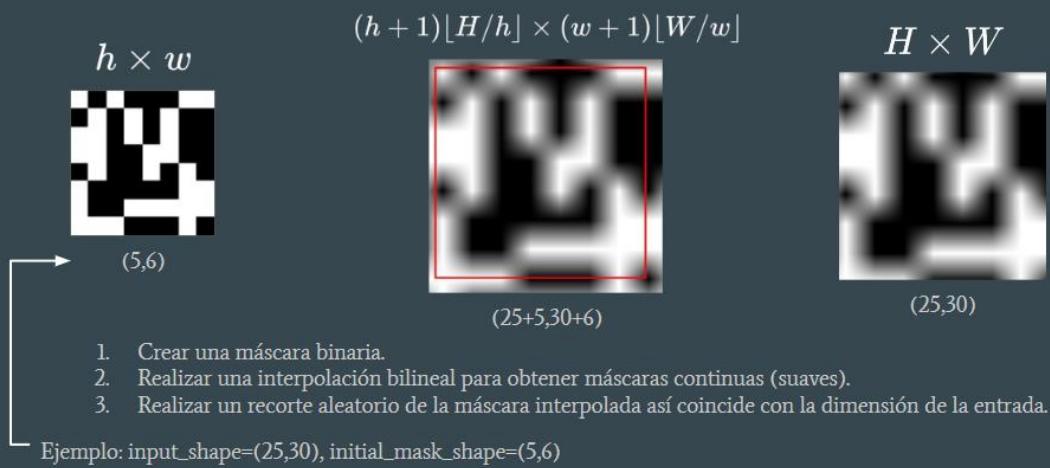
$$S_{I,f}(\lambda) \stackrel{MC}{\approx} \frac{1}{\mathbb{E}[M] \cdot N} \sum_{i=1}^N f(I \odot M_i) \cdot M_i(\lambda)$$

- Nota: nos quedamos solo con la **target class**



36

## RISE - Generación de Máscaras



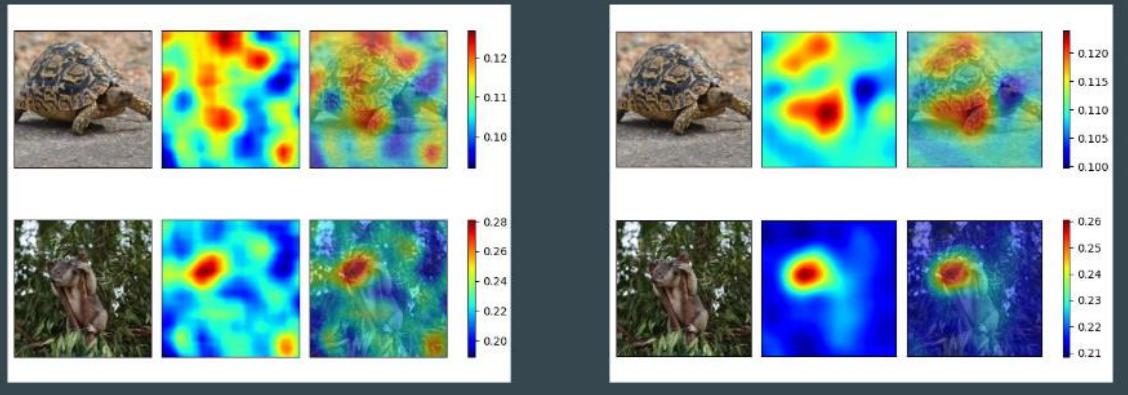
37

### Diferencias con Occlusion

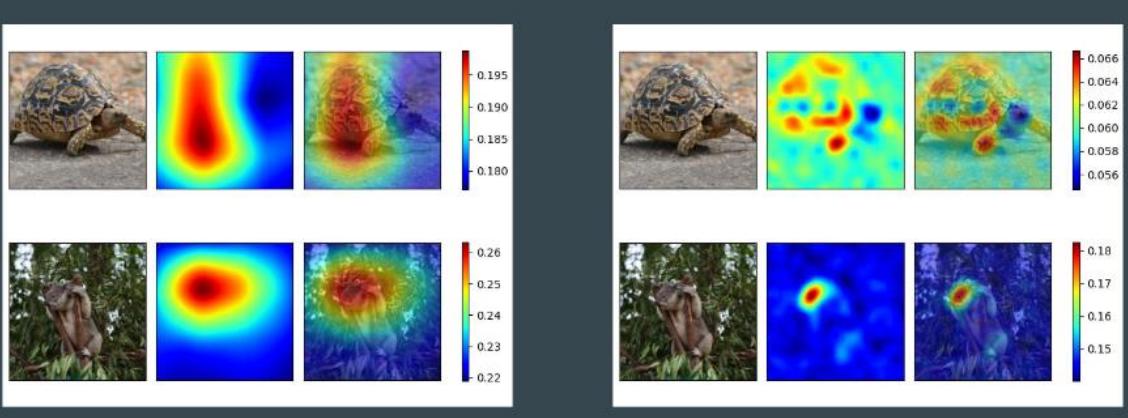
	Tipo de máscara	Tipo de ocultamiento	Uso de score de referencia
Occlusion	Máscaras binarias	Un parche	Sí
RISE	Máscaras continuas (más suaves)	Varios parches	No

## Sensibilidad a la cantidad de máscaras

- 128 vs 8192 máscaras, ambas de (8x8).



- (4x4) vs (15x15), ambas 8192 máscaras.

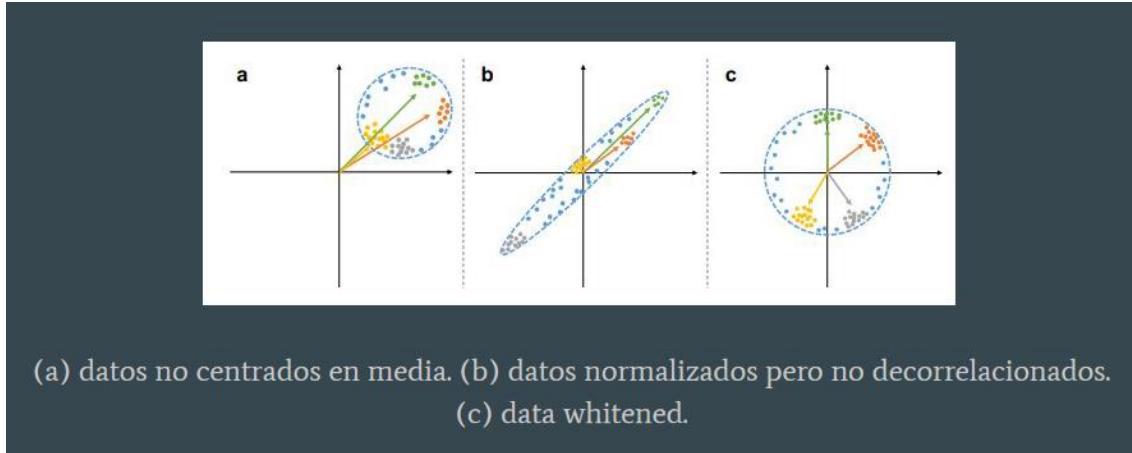


## Intrinsic Approaches

### Concept Whitening

- Método intrínseco
- Restringe/modela el espacio latente de un modelo obligándolo a que logre aprender a representar conceptos clave → usa vectores de concepto
- Los ejes del espacio latente se alinean con conceptos de interés conocidos
  - Implica que haya modificar la estructura de la red neuronal
- Principal Component Analysis (PCA) es una manera de reducir la dimensionalidad de los datos
- Whitening es también un paso habitual en el preprocesamiento que está relacionado con el PCA
  - Al trabajar con imágenes, es común que los datos sean redundantes ya que los píxeles adyacentes suelen estar correlacionados
  - El objetivo del whitening es hacer a la entrada menos redundante, es decir:
    - Los features de entrada (píxeles) están menos correlacionados
    - Todos los features de entrada tienen la misma varianza → Normaliza

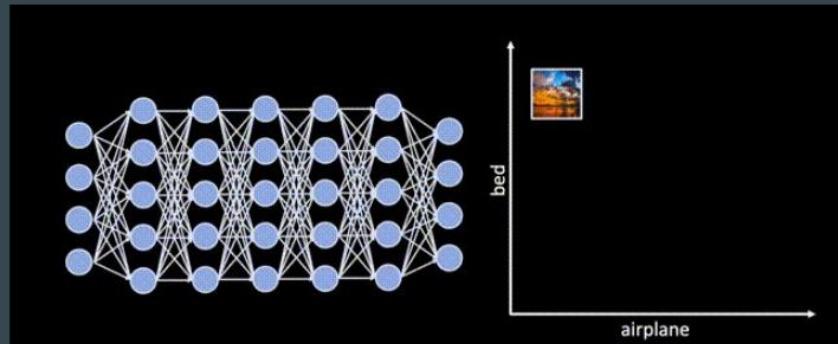
- Batch Whitening + Rotation Matrix (para transformación ortogonal)
  - o Se hacen por procesos de optimización iterativos alternados (gradientes)
- El primero deja los ejemplos con media cero, varianza unitaria y matriz de covarianza diagonal
  - o Generaliza mejor ya que “decorrelaciona y normaliza” el espacio latente
- El segundo alinea los conceptos con el eje en cuestión en direcciones ortogonales, de modo que desenreda los conceptos
  - o La matriz de rotación ajusta estratégicamente los conceptos a los ejes



- Al aplicar CW a una capa inferior tiende a capturar información de bajo nivel, como las características de color o textura de estos conceptos.
  - o La capa CW descubre características de un concepto más complejo.
  - o Si se aplica a una capa más cercana a la salida, se obtiene información de alto nivel.
- En la capa de la red donde se aplica CW los conceptos particulares se encuentran representado en un “eje” particular.
  - o Al examinar imágenes en este eje, se pueden extraer las características de “bajo o alto nivel” que la red utiliza para clasificar las imágenes como avión.
    - Ej. objetos plateados o blancos con fondo azul o grisáceo representan aviones

## Concept Whitening - Separabilidad del Espacio Latente

- Con los ejes alineados con conceptos predefinidos, se ve cómo la imagen recorre las capas de la red.

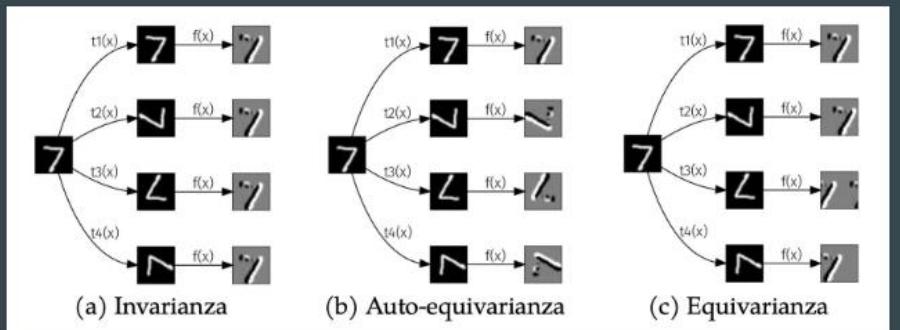


47

## Invarianza y Equivarianza

- Son propiedades de los modelos que permiten entender cómo estos se comportan ante cambios a su entrada. Dada una transformación  $t$  de  $T=[t_1 \dots t_m]$ , entonces:
- $f$  es **invariante** a  $t$  si:  $f(t(x))=f(x) \forall t \in T, x \in \text{Dom}(f)$ 
  - o Coloquialmente, una red es invariante a un conjunto de transformaciones si su salida no cambia cuando una de esas transformaciones se aplica a su entrada.
- $f$  es **auto-equivariante** a  $t$  si:  $f(t(x))=t(f(x)) \forall t \in T, x \in \text{Dom}(f)$ 
  - o Es una propiedad complementaria a la invarianza, de gran rigidez ya que requiere que  $t$  también esté definida para la salida de  $f$ , es decir,  $\text{Im}(f) \subset \text{Dom}(f)$ .
- $f$  es **equivariante** a  $t$  si:  $f(t(x))=t'(f(x)) \forall t \in T, x \in \text{Dom}(f)$ 
  - o Coloquialmente, una red equivariante a  $t$  si su salida cambia de forma predecible cuando  $x$  es transformado por  $t$ . Permite que la salida sea afectada de una manera controlada y útil.
  - o Formalmente, una función  $f$  es equivariante si existe una transformación  $t'$  que es la contrapartida de  $t$ , y que actúa en la salida de la función  $f$ .
  - o La equivarianza es una generalización de la invarianza y de la auto-equivarianza, y la invarianza es un caso particular de la equivarianza, donde  $t'$  es simplemente la función identidad ( $t'(x)=x$ ).

- (c) Transformaciones de rotación  $t$  de la entrada pueden corresponder a transformaciones de translación  $t'$  en la salida.



- Existen dos formas de dotar a los modelos con estas propiedades:
  - o Entrenar el modelo con aumentación de datos utilizando las transformaciones de interés induciendo al modelo a obtener tales propiedades.
  - o Modificar los modelos forzandolos a aprender estas características, ya sea cerca de la entrada o de la salida.
    - Spatial Transformer Network (STN) y Group Equivariant Convolutional Networks (GCNN).
- No está claro cuál es la mejor estrategia para impartir al modelo tales propiedades
  - o Se han definido métricas para analizar las redes neuronales desde una perspectiva transformacional

## Arquitecturas invariantes

### *Group Equivariant Convolutional Networks (GCNNs)*

- Las G-CNNs son una generalización natural de las CNN que reducen la complejidad del modelo al explotar simetrías de los datos de entrada.
  - o En imágenes, las simetrías podrían ser rotaciones o reflexiones.
- En una CNN tradicional, los filtros son invariantes a la translación. Esto significa que el mismo filtro se aplica en todo el espacio de entrada.
  - o Sin embargo, en muchos tipos de datos, como imágenes, hay otros tipos de simetrías que pueden ser explotadas.
  - o Ejemplo: en la tarea de reconocer un objeto en una imagen. No importa si el objeto está rotado o reflejado, sigue siendo el mismo objeto.
    - Las G-CNNs están diseñadas para reconocer el objeto sin importar su orientación

### GCNNs - G-convolutions

- Las G-CNNs extienden el concepto de invariancia a la translación a otros tipos de transformaciones, como rotaciones y reflexiones.
- Utilizan G-convolutions, un tipo de capa que posee de un grado sustancialmente mayor de compartición de pesos que las capas de convolución regulares.
  - o Aumentan la capacidad expresiva de la red sin aumentar el # de parámetros, incluso reduciéndolos, ya que el mismo filtro puede ser utilizado para diferentes orientaciones del mismo feature map.

- Estas funcionan aplicando filtros que son equivariantes a las transformaciones de cierto grupo.
  - o Esto significa que la respuesta del filtro cambia de la misma manera que la entrada cuando se aplica una transformación.
- Son utilizadas en campos que los datos tienen simetrías naturales (ej. imágenes).

### GCNNs – Ventajas

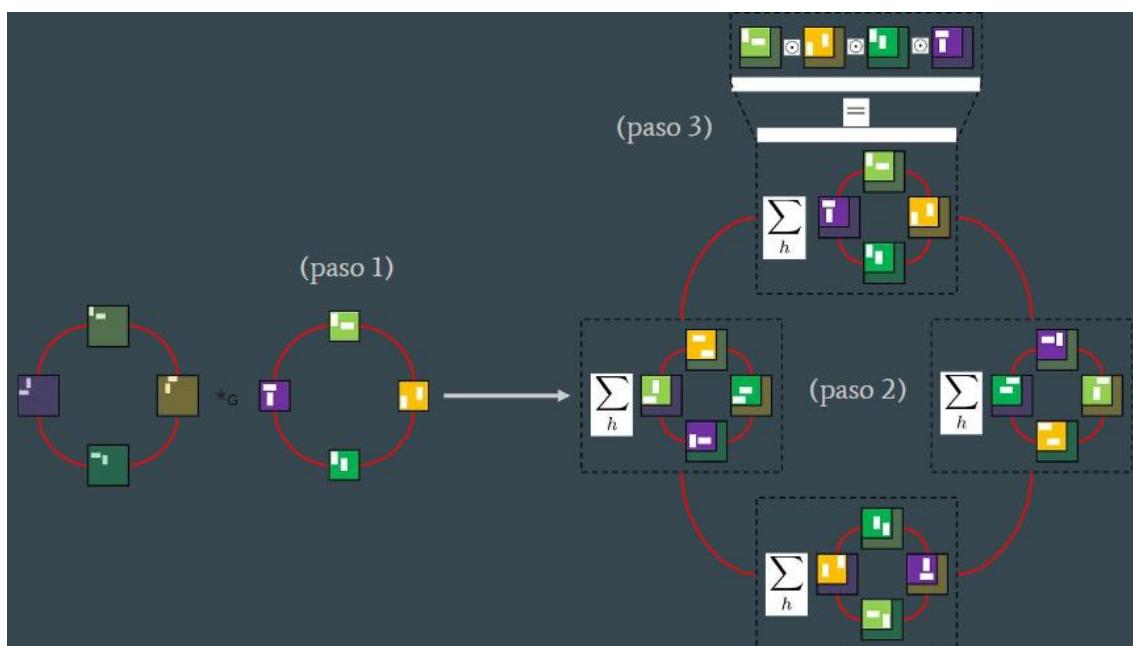
- Pueden mejorar la capacidad del modelo para generalizar a partir de datos limitados, ya que puede aprender a reconocer características en diferentes orientaciones sin necesidad de ver ejemplos de cada una.
  - o Esto además permite reducir la cantidad de datos de entrenamiento necesarios.
- Son fáciles de usar y se pueden implementar con un sobrecoste computacional despreciable para grupos discretos de traslaciones, reflexiones y rotaciones.

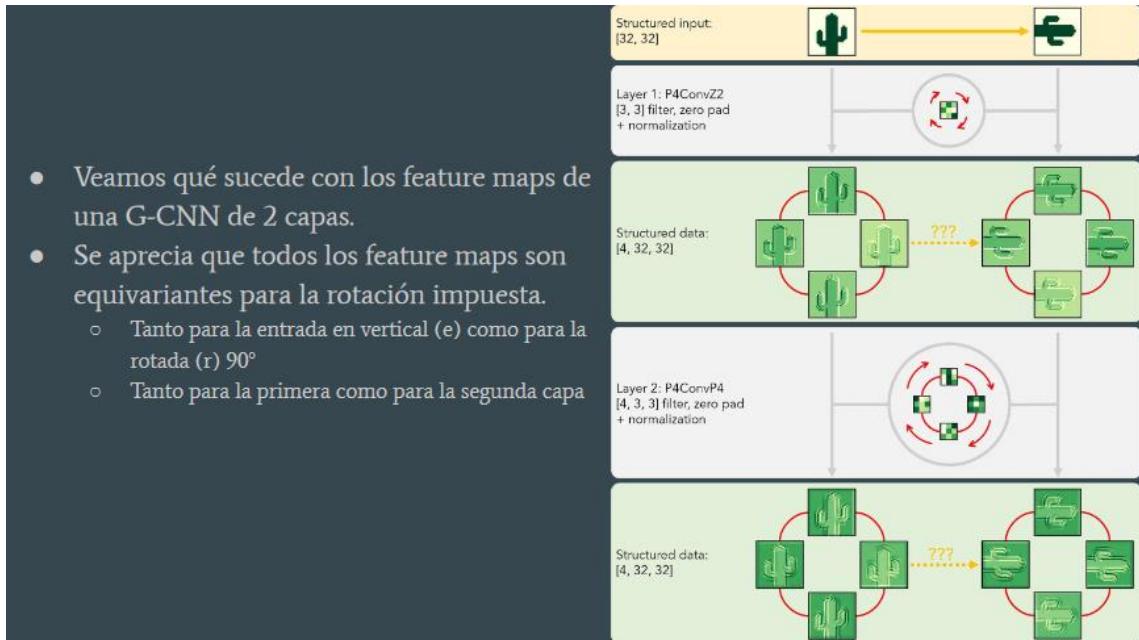
### GCNNs - G-convolutions (cont.) / Ejemplo (cont.)

El algoritmo completo de una G-convolución (en este la P4ConvP4) sería:

- 1- Creamos un filtro estructurado en el que en cada nodo los datos pueden ser independientes (el mismo utilizado en las dos ilustraciones anteriores).
- 2- Transformamos el filtro estructurado para cada elemento del grupo, creando así cuatro filtros estructurados transformados de manera diferente.
- 3- Tomamos por separado el producto punto entre estos filtros y el feature maps estructurado (entrada), creando así cuatro objetos de salida (no estructurados).
- 4- Finalmente, los asignamos en un feature maps estructurado de acuerdo a las transformaciones que el filtro estructurado tuvo que realizar.

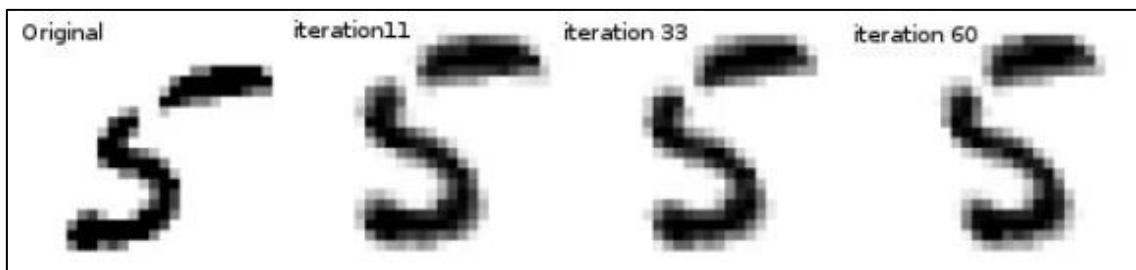
Así se crea un stack de feature maps ubicados de manera diferente que preserva la estructura del gráfico de grupos.





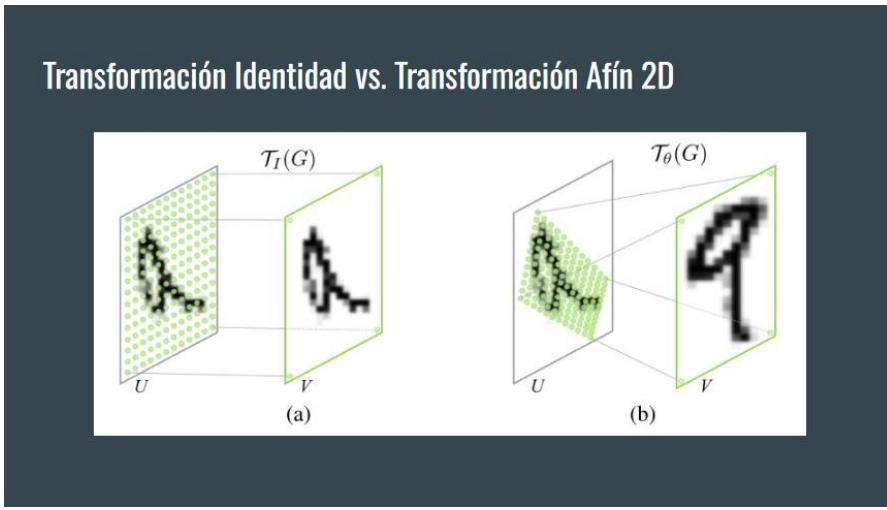
### Spatial Transformers Network (STN)

- Permiten a una red aprender cómo realizar transformaciones espaciales en una imagen de entrada para hacer que la red sea espacialmente invariante a los datos.
  - o Ej. recortar una región de interés, trasladar, escalar y corregir la orientación de una imagen.
- Las STN utilizan transformaciones afines para manipular activamente los feature maps espaciales, donde estas transformaciones son condicionadas por el FM.
  - o Es así como aprenden la invariancia a la traslación, escala, rotación y deformaciones más genéricas.



### Arquitectura

1. Localization Network: Calcula los parámetros de transformación  $\theta$  para aplicar cambios espaciales a la imagen de entrada. Es una CNN + RL al final.
2. Grid Generator: Produce una cuadrícula de coordenadas en la entrada basada en los parámetros de transformación  $\theta$ , según cada pixel de la imagen de la salida.
3. Sampler: Mapea la salida final a la entrada utilizando la interpolación bilineal a partir de las coordenadas generadas por el generador.



## Detección de objetos en imágenes

### Imágenes

#### Clasificación

- Consiste en determinar, dada una imagen, a qué categoría o clase pertenece a esa imagen.

#### Detección

- Consiste en “encontrar” dentro de una imagen, una subimagen, tal que sea clasificada en una categoría o clase determinada.
- Se utiliza para detectar uno o más “objetos” dentro de una imagen.

### Señales en general

#### Audio

- Clasificar instrumentos en solitario
- Detectar instrumentos en una orquesta

#### Video

- Clasificar videos
- Detectar escenas en videos

#### Señales de radar

- Clasificar una señal
- Detectar diferentes señales dentro de un aspecto

### Bounding box

**Bounding box:** alto y ancho de la caja donde buscar los objetos.

- Fuertemente dependiente del problema
- Una imagen puede contener el mismo objeto con diferentes tamaños.

**Stride:** indica la velocidad de desplazamiento del bounding box dentro de la imagen.

La opción de “**fuerza bruta**” consiste en deslizar uno o varios bounding box de diferentes tamaños por la imagen. Cada subimagen capturada se pasa por un clasificador para identificar el objeto (Con cierta probabilidad).



Imagen de 800x600. Bounding box de 80x60. Stride de 40x30.  
 $20 \times 15 = 300$  subimágenes

Los métodos multi-etapas son lentos y computacionalmente costosos

- Imposibles de usar para detección en tiempo real.

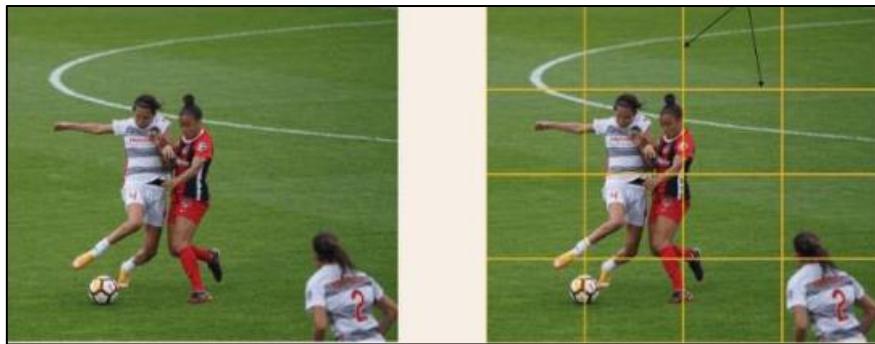
Alternativamente se pueden utilizar los métodos conocidos como single-shot object detectors.

- Detectan varios objetos al mismo tiempo
- Predicen bounding boxes (además de la clase del objeto)
- Son robustos para su uso en sistemas de tiempo real
- SSD (Single Shot MultiBox Detector)
- YOLO (You only look once)

## YOLO (You Only Look Once)

Es una red neuronal convolucional

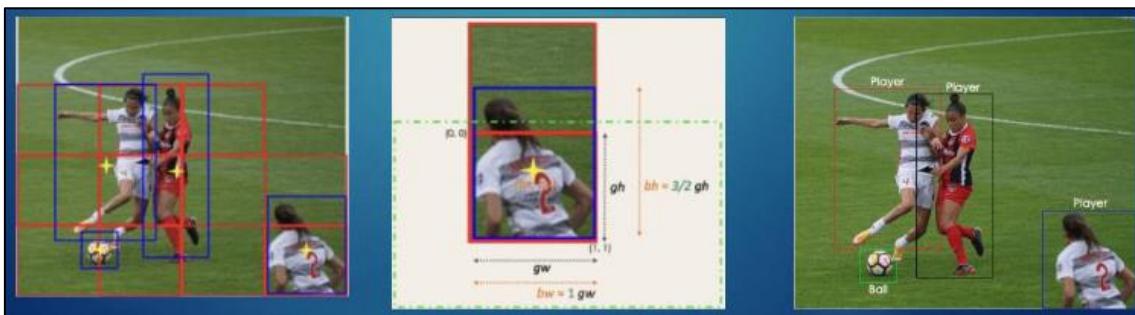
Divide la imagen en NxN cuadrantes del mismo tamaño



El objetivo es que cada cuadrante identifique el objeto que está en su interior.

Cada cuadrante predecirá un objeto con su posible bounding box. La predicción tendrá:

- La probabilidad de contener un objeto (pc)
- El centro del bounding box (bx, by)
- El largo y ancho del bounding box (bh, bw)
- La probabilidad de que el objeto sea de una clase específica.



## Función de error

YOLO utiliza varias métricas de error:

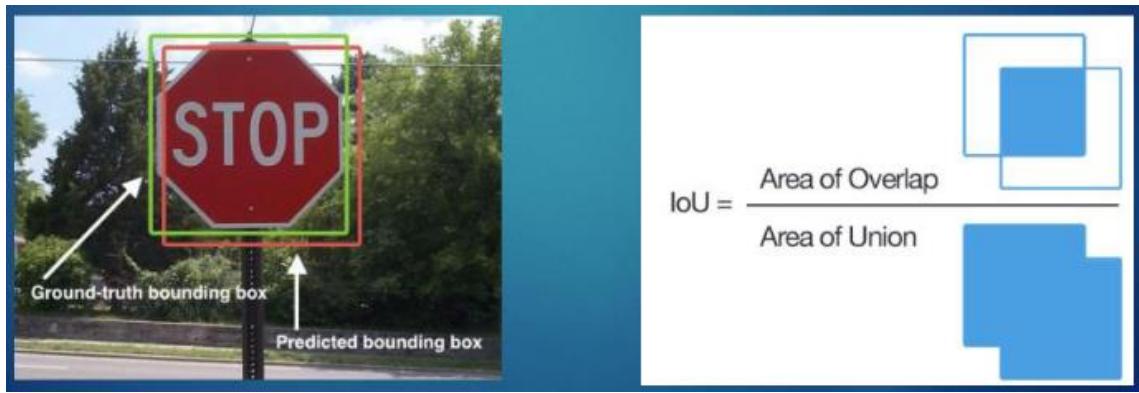
- El error de clasificación (de la clase del objeto)
- El error de localización (entre la bounding box predicha y la real).
- El error de confianza (en haber detectado un objeto).

El error total es la suma de todos los errores.

## Métricas de confianza

YOLO utiliza la métrica conocida como Intersection over Union (IoU) para evaluar cuando una predicción se corresponde a un bounding box real.

Luego utiliza un umbral (típicamente 0.5) para catalogar la IoU como TP o FP.



¿Cómo tenés el bounding box real? (preguntar)

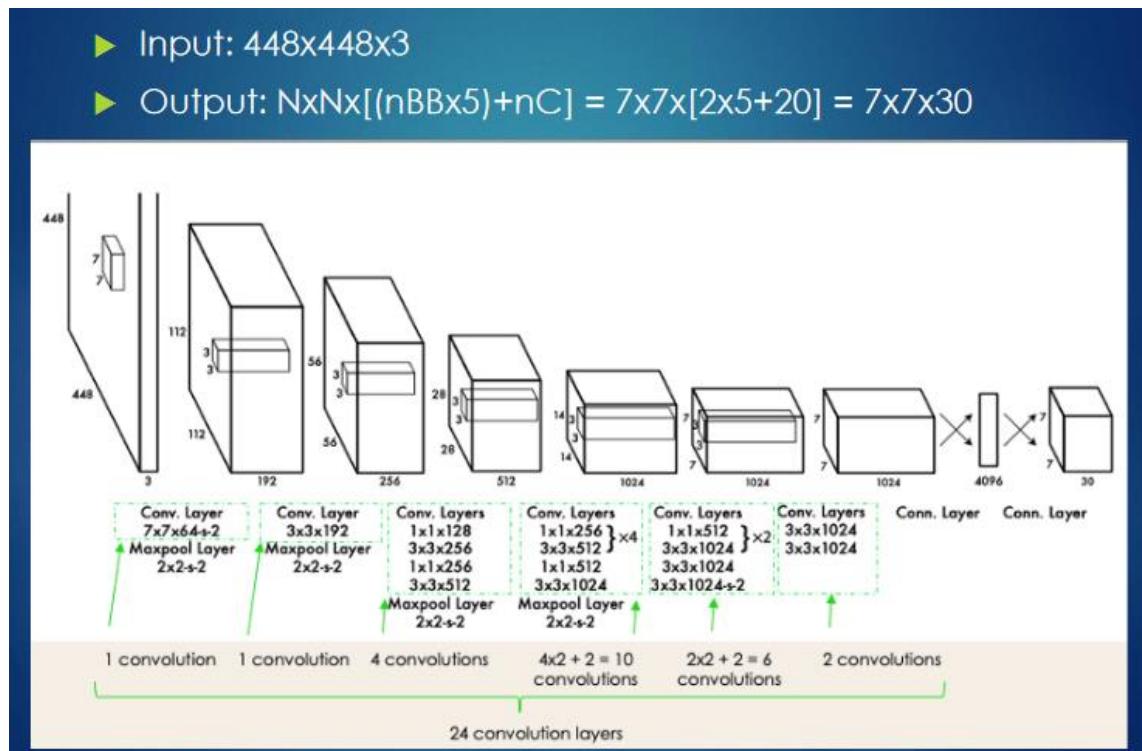
## Limitaciones

YOLO suele tener dificultades cuando se encuentran muchos elementos en el mismo cuadrante.

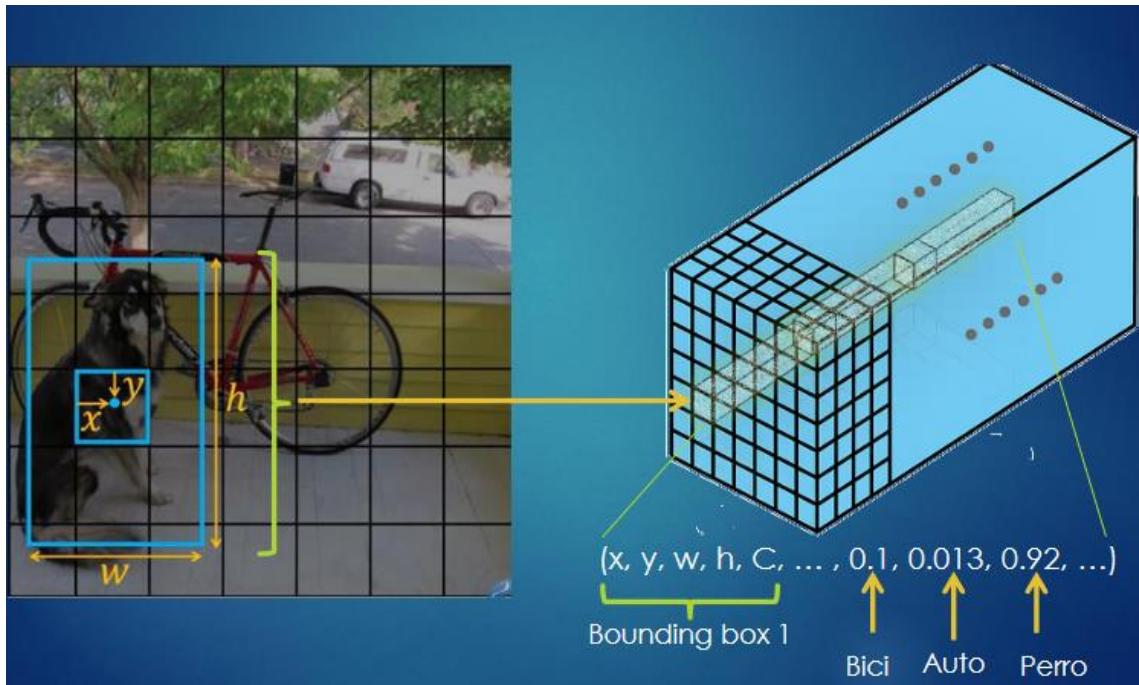
- Por lo general detecta hasta dos, aunque puede aumentarse.

YOLO solo sirve para detectar objetos de forma rectangular, por lo que no sirve para detectar formas más específicas o a través de cámaras como ojos de pez.

## Arquitectura

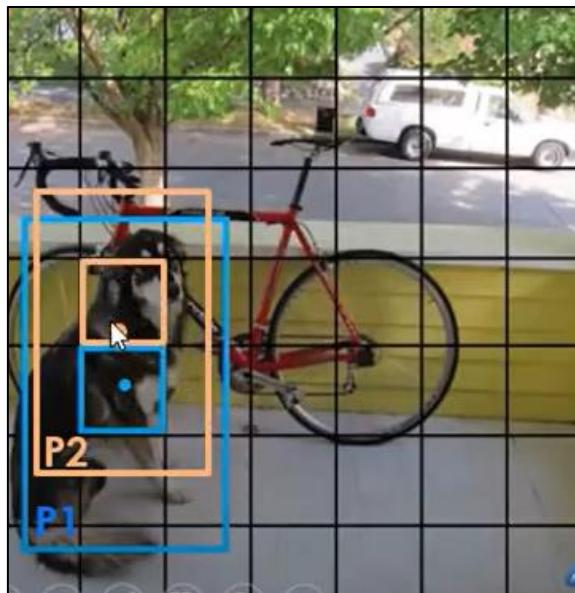


Interesante: los píxeles en donde se encuentra el objeto (en ejemplo, el cuadrado de 7x7), al realizar las convoluciones se va a ubicar espacialmente en el mismo cuadrante del feature map de salida



## Eligiendo la mejor respuesta

YOLO usa el método denominado “non max suppression” para elegir la mejor respuesta ante un mismo objeto.



Es un método post-procesamiento, que trabaja con la salida de la red.

Si  $\text{IoU}(P1, P2) > \text{umbral} \rightarrow P = \text{argmax}(C(P1), C(P2))$

## Usos

- Sistemas de vigilancia
- Detección de productos con fallas
  - o Detección durante la producción
- Robótica
  - o Obstáculos para sortear

- Objetos para agarrar
- Imágenes médicas
  - Tumores u otras patologías
- Reconocimiento de estructuras dentro de documentos
  - Párrafos, figuras, tablas, etc.
- Detección de espectros en placas espectrográficas
- Detección de manos en imagen
- Detección de semillas de trigo
  - Semillas buenas y dañadas

## YOLO en video

Para detectar objetos en video el procedimiento es el mismo que vimos anteriormente.

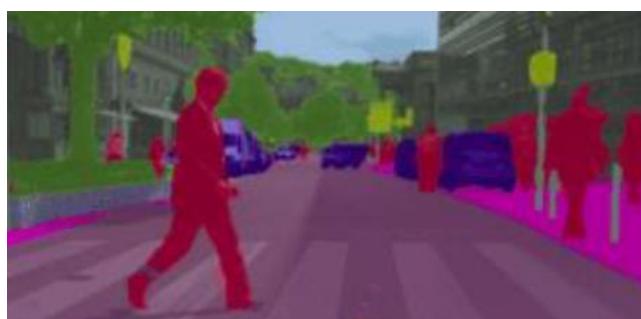
Cada frame capturado en tiempo real (una imagen) es pasado por el modelo para que detecte los objetos en la imagen. Esta captura puede ser utilizada como frame para construir el video de salida (video original + bounding boxes)

## Segmentación de imágenes

La segmentación de imágenes es una técnica que busca obtener, a partir de una imagen digital, distintos grupos de píxeles (segmentos de imagen).

- Facilita la detección de objetos

Los resultados de la segmentación de imágenes forman máscaras de segmentación, que representan el límite y la forma específica, píxel a píxel, de cada objeto o región dentro de la imagen.



Hay tres tipos de tareas:

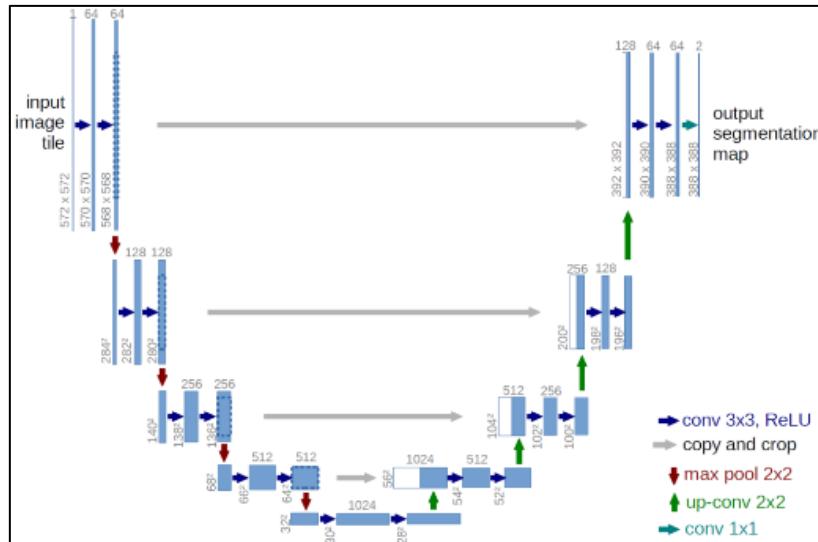
- Segmentación semántica
  - Asigna una clase semántica a cada píxel
  - Objetos contables como "auto", "árbol" o "persona".
  - Objetos "incontables" como "cielo", "agua" o "pasto".
- Segmentación de instancias
  - Es una forma de detección de objetos que genera una máscara de segmentación en lugar de un cuadro delimitador.
    - En vez de detectar "personas", detecta "persona1", "persona2"
- Segmentación panóptica
  - Es una combinación de las dos anteriores

## U-NET-Arquitectura

Es una red totalmente convolucional.

Es de la familia de las redes auto-encoders.

- Consta de un codificado y de un decodificador.



El tamaño de la salida es igual a la de la entrada.

Tiene un codificador, un decodificador y “puentes” que permiten mantener el espacio latente de la imagen y construir una imagen representativa. Por ejemplo, si se busca una imagen que tenga tres personas, el decodificador necesita ganar información de dónde estaban esas personas.

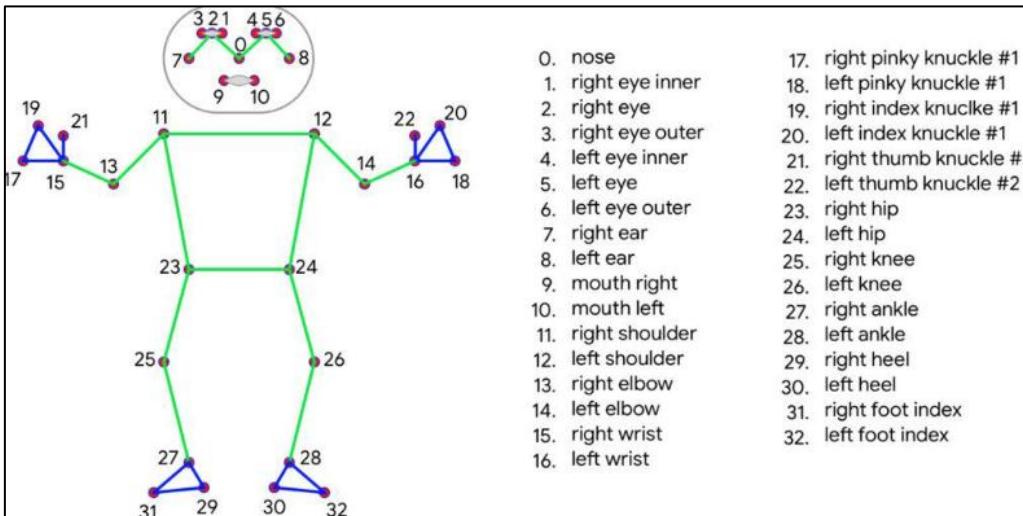
La red aprende a “cambiar de color” a los píxeles del mismo objeto.

### Usos

- Diagnóstico médico
  - o U-Net: Convolutional Networks for Biomedical Image Segmentation
- Conducción autónoma de vehículos
- Cartografía de imágenes satelitales
- Agricultura de precisión

## Detección de poses

La detección de la pose de una persona consiste en detectar los puntos de las articulaciones (cabeza, hombros, codos, manos rodillas, cadera, pies, etc.) y unirlos para formar un “esqueleto”.

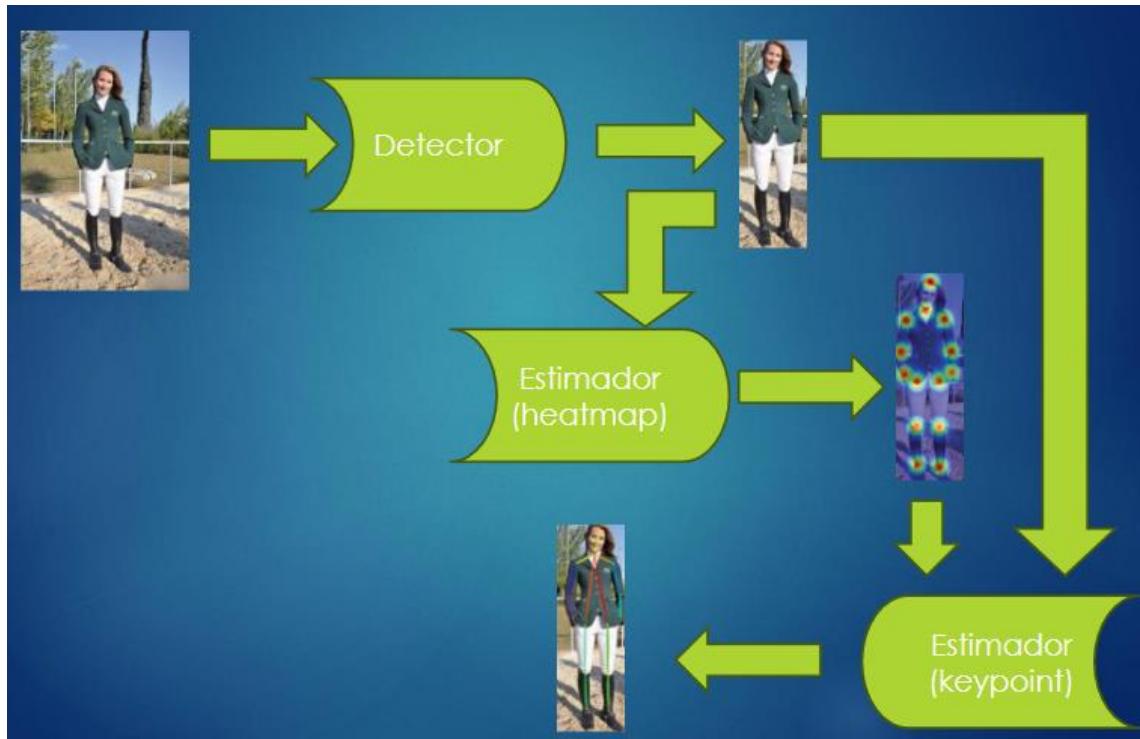


## BlazPose

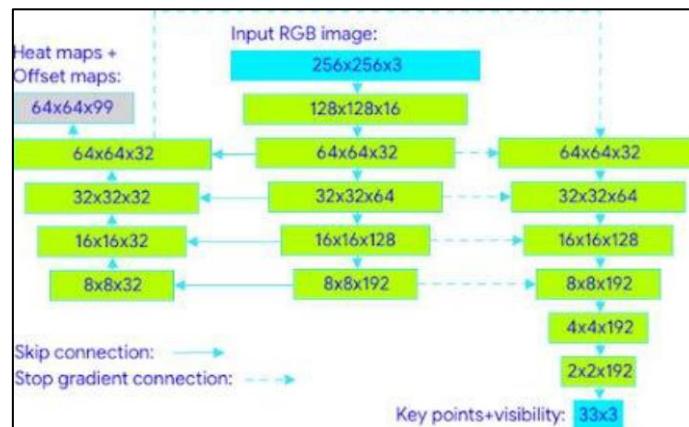
### Arquitectura

Consiste en dos modelos:

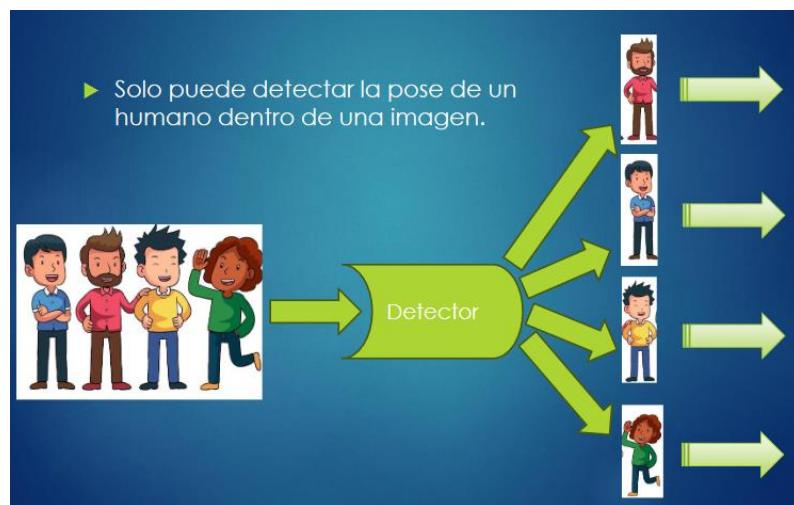
- Detector. Su objetivo es obtener el bounding box de la persona en la imagen.
  - o La salida es un único bounding box, el cual puede estar rotado.
- Estimador. Dada la subimagen devuelta por el detector, estima la mejor posición de los keypoints.
  - o A su vez, este modelo consta de dos submodelos
    - Una primera parte que detecta los “heatmaps”
    - Una segunda parte que usa la imagen original y el heatmap aprendido para estimar los keypoints
  - o Los 33 keypoints devueltos son puntos 3D



### Estimador



### Limitaciones

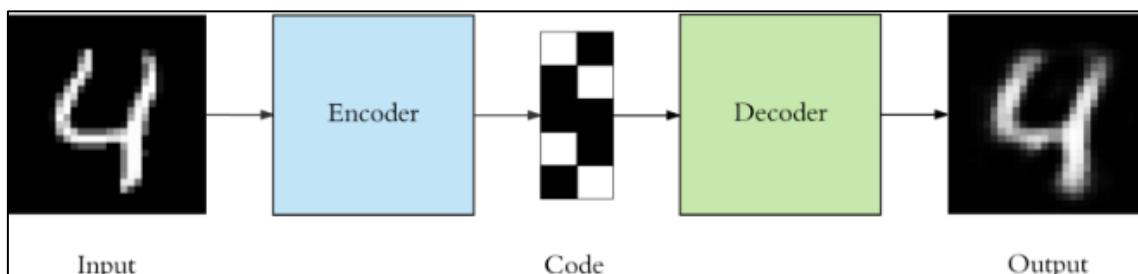


## Usos

- Fisioterapia
- Entretenimiento con realidad virtual y aumentada
- Interacción humano-computador
- Reconocimiento de gestos

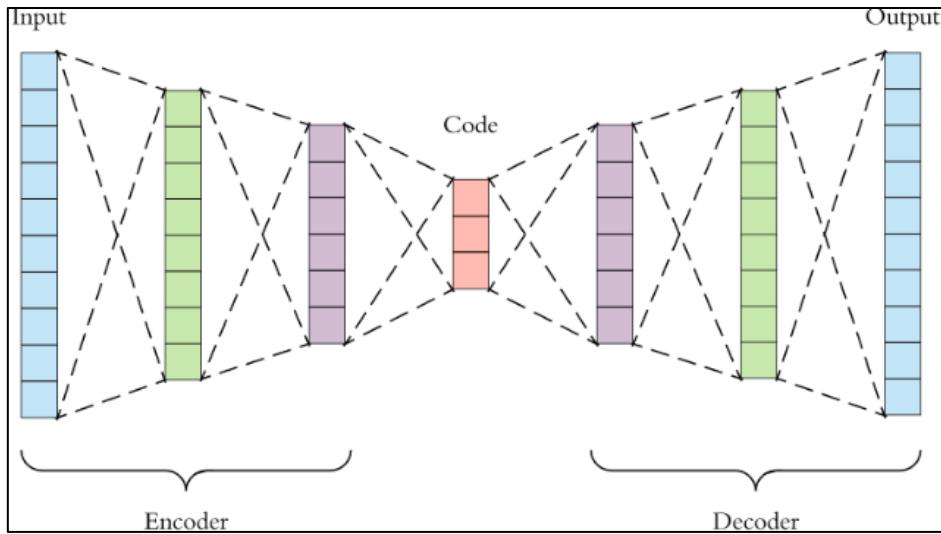
## Autoencoders

- La idea de los autoencoders nace en el año 1986.
- Los primeros trabajos lo usaron para detectar locutores y para compresión de imágenes.
- En 1998 algunos investigadores encontraron serios inconvenientes y se las dejó de lado.
- A partir del año 2006 se volvieron muy utilizados con el nacimiento del movimiento denominado "Deep learning", específicamente para la compresión de imágenes.
- Actualmente se los utiliza para la generación de imágenes
- Las redes autoencoders son redes de aprendizaje "no-supervisado".
- El objetivo es entrenar un modelo que sea capaz de reconstruir la entrada.
- Intentan emular la función identidad
- Son utilizadas para la compresión de datos.
- La red se entrena presentando un ejemplo  $X$  para obtener una salida  $X'$ . Esta salida es comparada con el propio  $X$ . Se espera que la salida de la red sea la misma que la entrada.
- Cuando el número de neuronas en la capa oculta es menor al número de neuronas en la capa de entrada al autoencoder se lo denomina undercomplete.
  - o Al entrenar este tipo de autoencoders estamos forzando a aprender las características más relevantes de los datos de entrenamiento.
    - Se construye el espacio latente
- Cuando el número de neuronas en la capa oculta es mayor al número de neuronas de la capa de entrada al autoencoder se lo denomina overcomplete.
  - o Este tipo de autoencoders permite aprender características topológicas de los datos de entrenamiento.
- Para construir un autoencoder se necesita un método de codificación (encoder) y otro de decodificación (decoder).
- La salida del codificador que sirve como entrada al decodificador se lo conoce con el nombre de "code" o "vector latente" o "embedding".



## Autoencoders multicapa

- Tanto el encoder como el decoder pueden tener cualquier profundidad
- Por lo general se construyen **espejadas**



Al momento de presentar un ejemplo  $X$ , se compara la salida obtenida ( $X'$ ) con la propia entrada ( $X$ ). El error cometido se utiliza para ajustar los vectores de pesos de las neuronas.

$$e(X) = \sum_{i=1}^n (X_i - X'_i)^2$$

Cada ejemplo es reconstruido con un cierto error.

- Existen varias formas de medir el error.
  - o Error cuadrático medio
  - o Entropía binaria si los valores están entre 0 y 1.

## Entrenamiento

Dado un patrón  $X$

Se calcula el code:

$$Z = \text{sig}(\Sigma(XW_e + b_e))$$

Se "decodifica" el code en la capa de salida:

$$X' = \Sigma(ZW_d + b_d)$$

Se calcula el error de reconstrucción

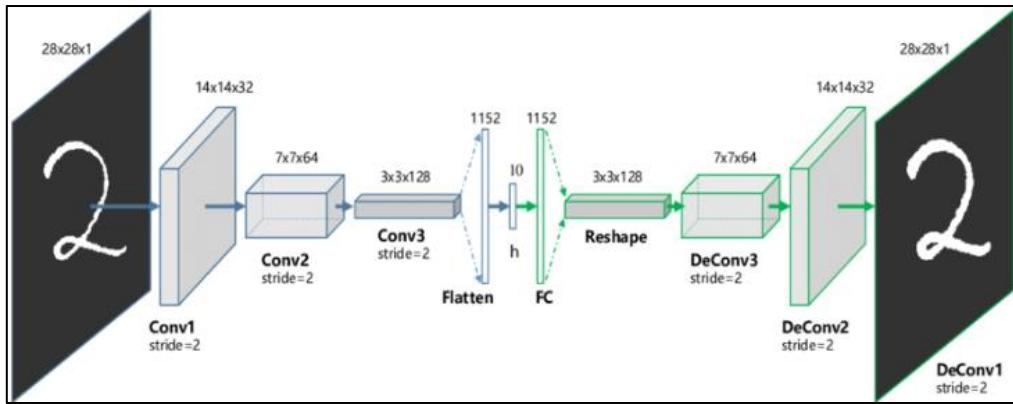
$$E = |X - X'|$$

Los pesos se actualizan utilizando el algoritmo de backpropagation y la técnica del gradiente.

## Generando imágenes con autoencoders

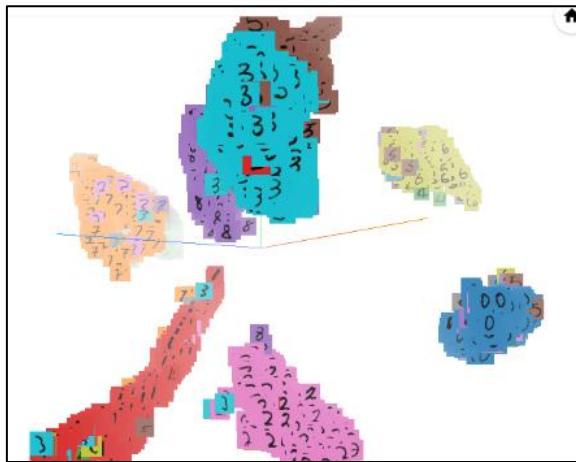
En el dominio de imágenes el encoder se arma con capas convolucionales y poolings.

El decoder con capas de upsamplings.



## Espacio latente

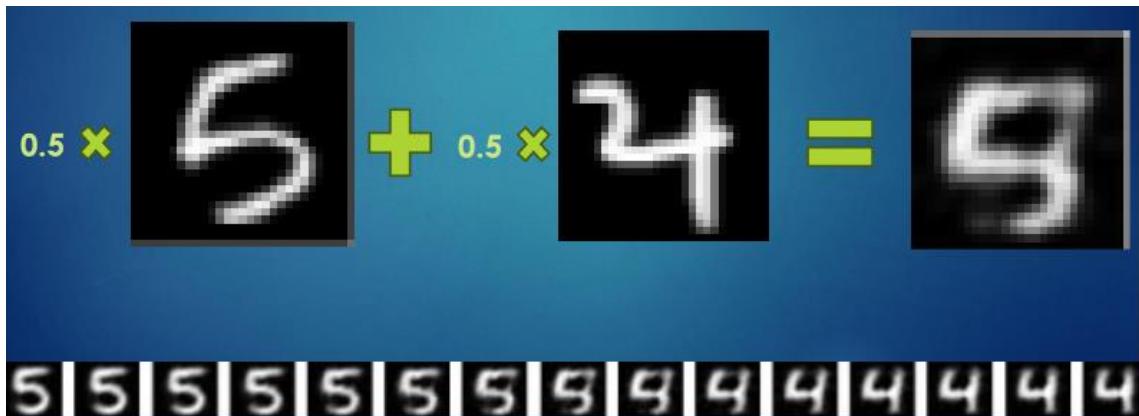
Los embeddings del dataset forman lo que se conoce como **espacio latente**



Tener un decoder que genere imágenes a partir de embeddings permite generar una imagen a partir de un nuevo embedding.

## Vector medio

Podemos pedirle la imagen correspondiente al vector medio entre dos vectores.

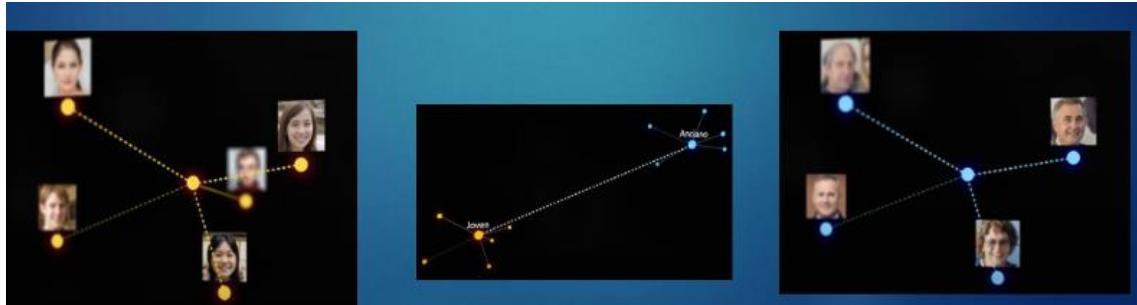


## Modificando el embedding

Podemos obtener el vector medio para conjuntos particulares.

- Ej: personas jóvenes y ancianas

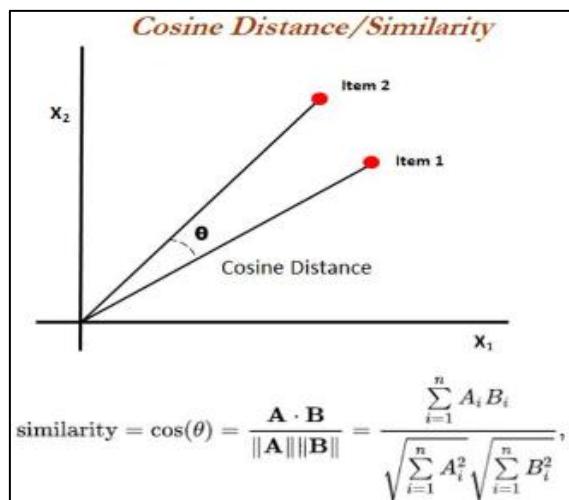
Luego, a partir de un vector de una cara en particular, podemos movernos hacia uno de esos centros para obtener versiones más jóvenes o más ancianas.



## Búsqueda de imágenes similares

A partir del embedding de una imagen podemos encontrar los embeddings más cercanos.

- Por ejemplo, usando la similitud coseno



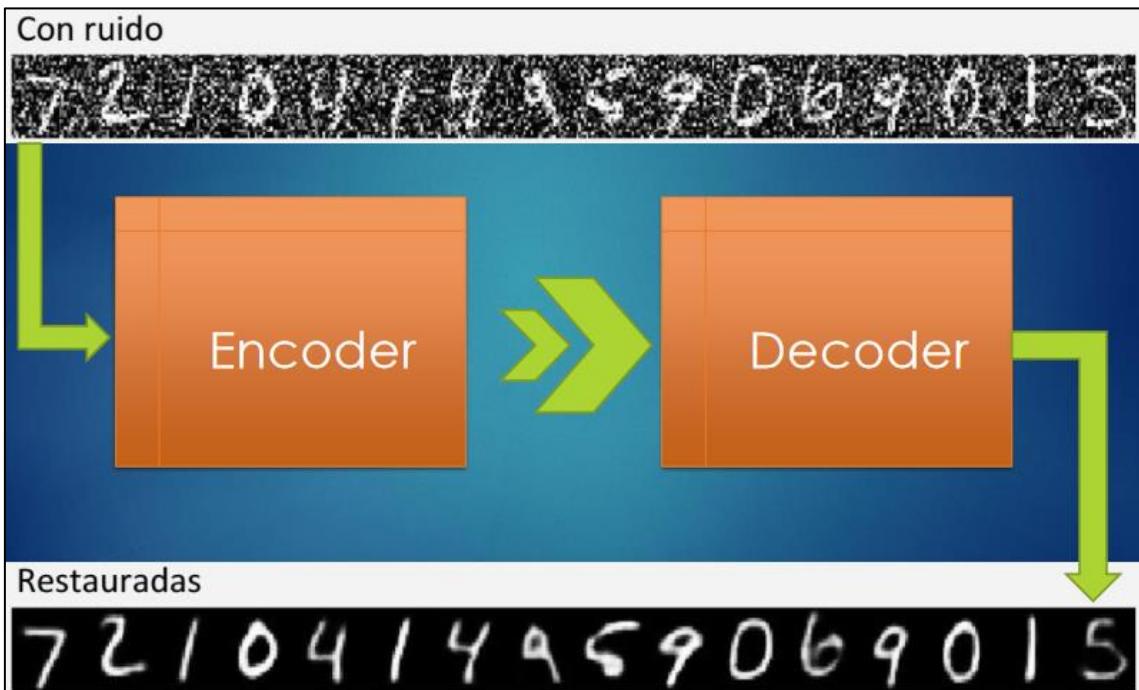
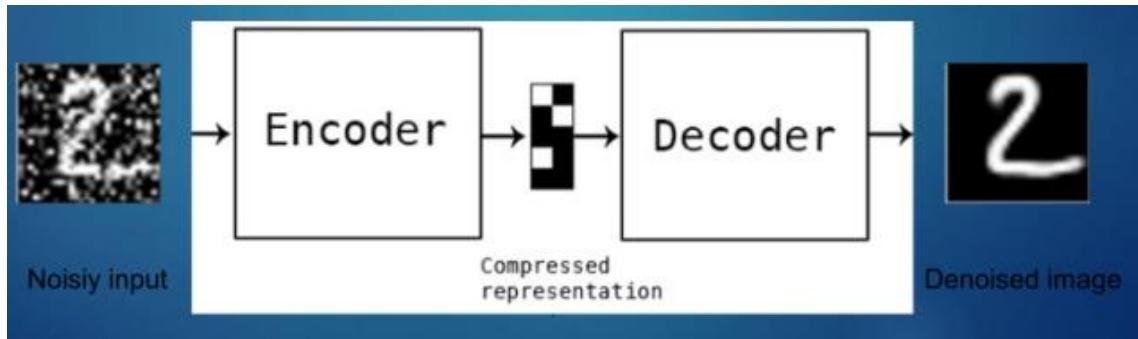
## Sparse autoencoders

- Uno de los problemas de los autoencoders es que pueden conseguir un alto grado de overfitting de manera muy fácil.
  - o Se comportan como una verdadera función de identidad (lo cual no siempre es lo deseado).
- Es necesario que estas arquitecturas aprendan las características estadísticas de los datos de entrenamiento.
- La clave es usar un sistema de penalización que pueda aprovechar estas características.
- Los llamados sparse autoencoders utilizan un mecanismo de regularización.
- Un mecanismo restringe a solo una fracción de las neuronas de la capa oculta se "activen" arrojando como salida un valor distinto de cero.
  - o Este mecanismo consiste en penalizar la salida de las neuronas ocultas.
  - o Se busca que solo un subconjunto de estas neuronas se "especialicen" en ciertas características de los datos de entrada.
- Otro mecanismo consiste en agregar una regularización al error de la red.

$$L_{sparsity}(\theta, \phi) = \mathbb{E}_{x \sim \mu_X} \left[ \sum_{k \in 1:K} \omega_k ||h_k|| \right]$$

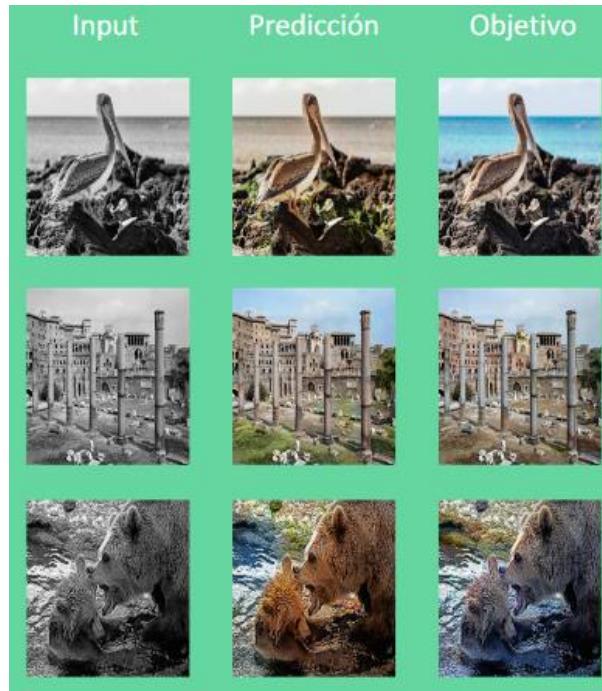
## Denoising autoencoders

- El objetivo de este tipo de autoencoders es conseguir aprender las características más relevantes de la entrada, para que permitan una buena reconstrucción.
- Se entrena de la misma manera que los auto-encoders convencionales.
- Se utilizan para eliminar ruido



## Generación de color

Se puede entrenar un modelo para realizar esta tarea simplemente generando un dataset de imágenes en blanco y negro como inputs y sus respectivas imágenes a color como slida.



## Superrresolución

- Se trabaja con un dataset de imágenes de menor calidad que las originales (como inputs).
- En este caso, la salida tendrá que tener un tamaño mayor a la entrada.



## Variational autoencoders

- Los Variational AutoEncoders (VAE) son modelos de aprendizaje que mezclan las redes neuronales con distribuciones de probabilidad.
- Los Variational Autoencoders aprenden la distribución de probabilidad de la entrada, funcionando como modelos generativos.

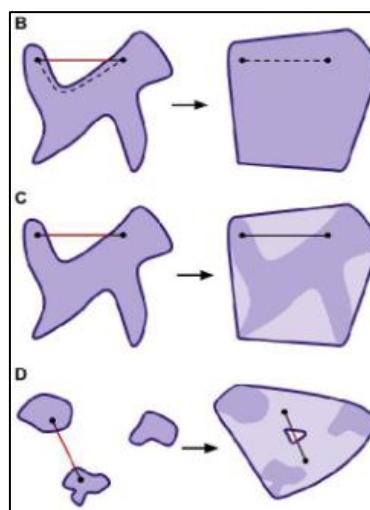


## Espacio latente

El principal problema que se ha constatado es que, en la mayoría de los casos, las representaciones intermedias que se obtienen forman un espacio que no es continuo, sino formado por diversas “bolsas” aisladas que agrupan en su interior representaciones de datos de entrada similares (clases)

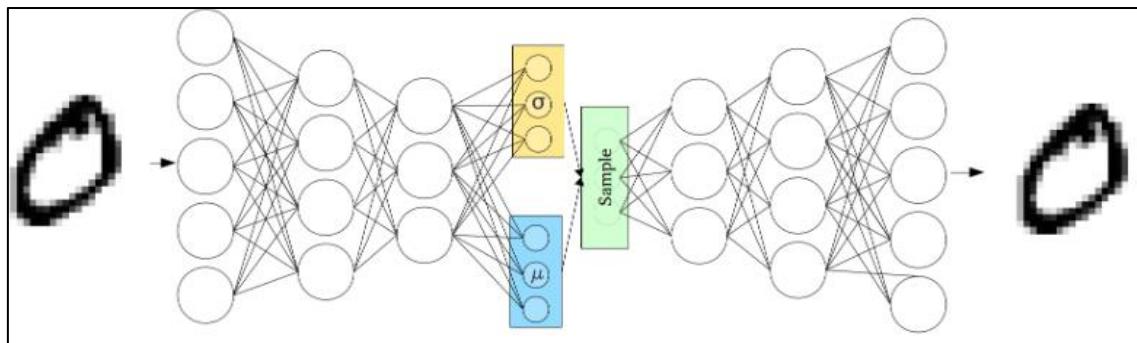
Una interpolación en el espacio de representaciones no se corresponde con una interpolación similar en el espacio original de datos.

Una muestra en una zona intermedia producirá una salida muy poco realista. Esto es porque el decoder no tiene información acerca de cómo manejar una representación que proviene de esa región del espacio latente



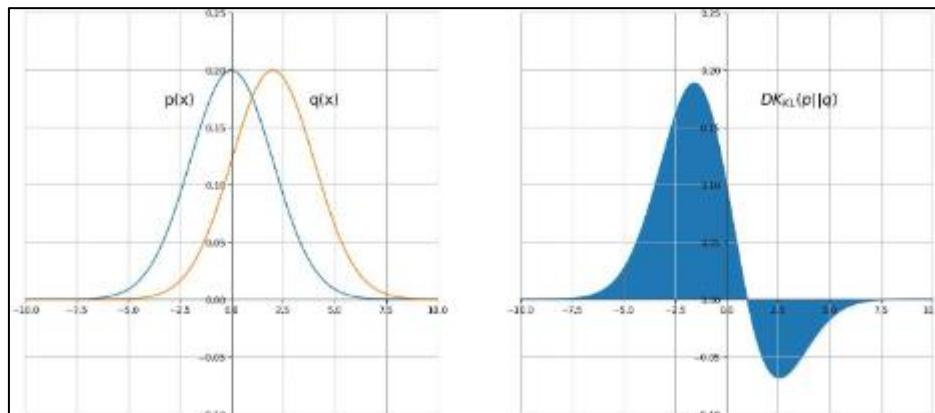
## Espacio latente

- En un VAE el espacio latente “probabilístico” está formado por dos vectores de igual longitud.
  - o un vector de medias
  - o un vector de desviaciones estándar
- A partir de este espacio latente “probabilístico” se genera un vector latente “discreto”



## Función de error

- Las métricas ECM, MAE y entropía no sirven por si solas para entrenar un VAE.
- Se utiliza la métrica llamada KL-divergencia (Divergencia de Kullback-Leibler) y el método de máxima verosimilitud.
  - o Mide la diferencia existente entre dos distribuciones de probabilidad.
- El error del modelo puede calcularse como:
  - o Error = ECM + KL



## Optimización de modelos probabilísticos

- Dado un conjunto de datos  $D = \{x(1), \dots, x(N)\}$  donde  $x(i)$  son muestras pertenecientes a una distribución subyacente.
- El modelo paramétrico que aproxima la distribución conjunta de probabilidad, será de la forma  $p_\theta(D) = \prod_{i=1}^N p_\theta(x_i)$
- La forma típica de calcular  $\theta$  es optimizar esta función, esto es, la estimación por máxima verosimilitud (MLE) de las instancias pertenecientes al conjunto de datos  $D$ .
- En VAEs el método MLE equivale a la minimización de la divergencia de Kullback-Leibler entre la distribución de los datos y la del modelo.
- Disponiendo de la función de optimización MLE ya es posible calcular los parámetros  $\theta$  de la red neural utilizando los métodos de entrenamiento convencionales, por ejemplo, el descenso de gradiente estocástico.

## Resumen

- Aprendizaje “no supervisado”
- Salida esperada igual a la entrada

- Usos
  - o Búsqueda de imágenes
  - o Compresión de imágenes
  - o Reducción de la dimensionalidad
  - o Modelado de tópicos

## Redes generativas adversarias

“La capacidad de crear”

Es una arquitectura formada por dos redes.

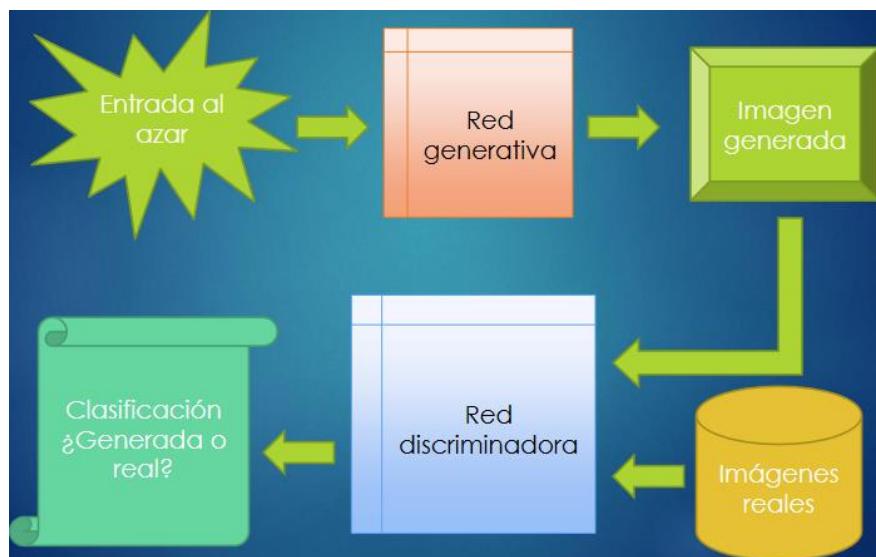
- Una red generadora de datos
- Una red discriminadora de datos

El objetivo de estas redes es conseguir que un cierto dato logre adoptar la distribución de otro conjunto de datos.

Su mayor uso es el de generar nuevos datos

- Imágenes
- Audio
- Texto

## Arquitectura



### Red generadora

Imposible de entrenar por si sola

- ¿Cuál es la métrica de error?
- La red discriminadora provee el gradiente de error

Su objetivo es llegar a producir una imagen tal que la red discriminadora diga que es real con una confianza alta.

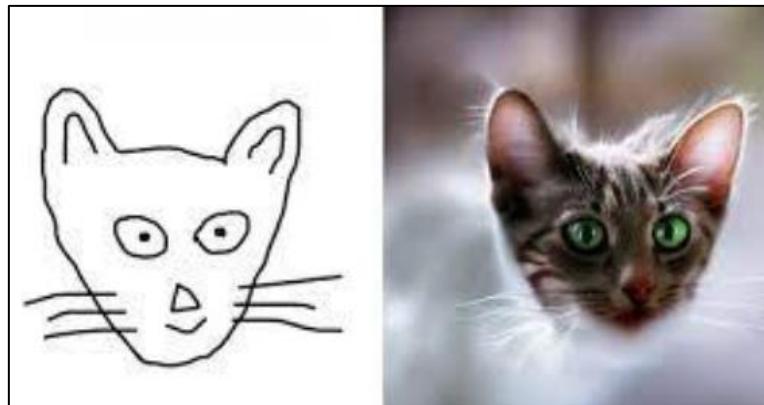
Aprendizaje “no-supervisado”

## Red discriminadora

Se la entrena con imágenes reales y con imágenes generadas (clasificación)

- Se debe contra con una base de datos adecuada al dominio

Aprendizaje supervisado



## Entrenamiento

- Generar K imágenes con la red generadora
- Seleccionar K imágenes reales
- Entrenar red discriminadora
  - o Colectar errores de imágenes generadas
- Entrenar red generadora para disminuir el error
  - o Imagen generada con score “generada” → producción mala
  - o Imagen generada con score “real” → producción buena
- Se debe lograr un buen equilibrio entre ambas redes
  - o Si la generadora logra engañar fácilmente a la discriminadora, cualquier imagen será interpretada como imagen real.
  - o Si la discriminadora aprende muy bien su trabajo, la generadora no tendrá chance de engañarla.
- Finalizado el entrenamiento, la red generadora será capaz de crear imágenes “reales”



## Función de error

- Se busca optimizar una función objetivo que “mida” ambos modelos a la vez.
- A grandes rasgos, es como una función de Entropía cruzada (inversa) que el generador tratará de minimizar y el discriminador tratará de maximizar

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))].$$

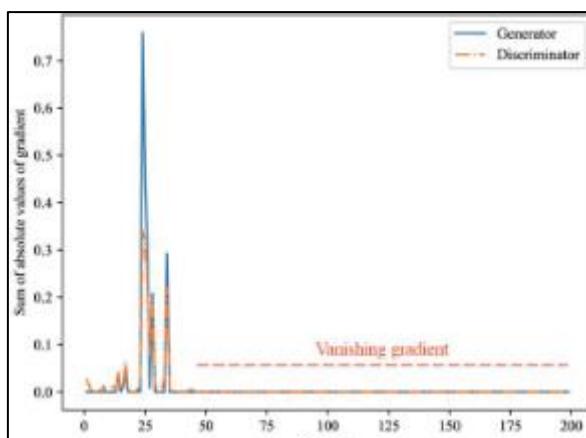
- Dada la naturaleza de la función de error, no es trivial definir cuándo detener el entrenamiento:
  - o Tratar de igualar el error de ambos modelos (esto se repetirá varias veces a lo largo del entrenamiento)
  - o Usar métricas definidas para el problema de generación de imágenes:
    - Inception Score
    - Fréchet Inception Distance (FID)

## Colapso de moda

- Un problema que no fue previsto de entrada con las redes GAN fue que el generador podía aprender a generar un conjunto muy chico de salidas o modas.
- Se esperaba que G genere muestras distintas para cada vector aleatorio. Pero, en cambio, puede suceder que busque siempre una única salida que funcione para engañar a D.
  - o Puede verse como un sobreajuste a D.

## Desvanecimiento del gradiente

- Si D es demasiado bueno, G podría fallar ya que un discriminador óptimo no otorga suficiente información a G para progresar en su entrenamiento.
- No se actualizan más los pesos de D y las actualizaciones a G no son efectivas.
  - o A esto se lo conoce como desvanecimiento de gradiente (vanishing gradient)



## Usos

- Aumentación de datos
- Educación

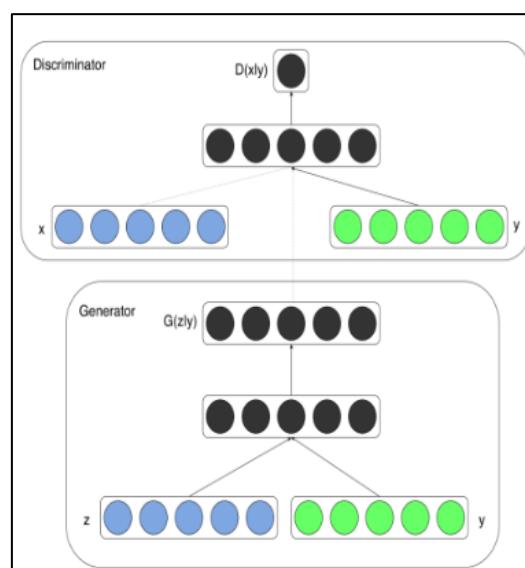
- Generador de ejercicios
- Corrector
- Generación de código fuente
- Generación de imágenes, audio y video
- Generación de música
- Asistentes virtuales

## Generación de imágenes condicional

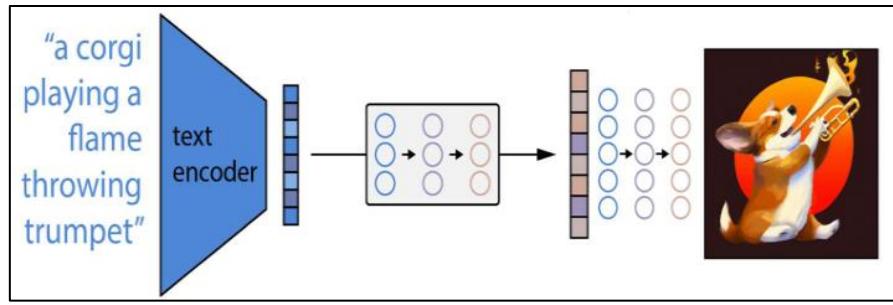
¿Cómo se puede tener más control acerca de los datos que se desean generar?



- Para lograr esto hace falta agregar información de la clase como entrada del modelo generador
- Esto es posible simplemente concatenando el correspondiente vector one-hot.
- En modelos tipo GAN, la información de la clase se debe incluir tanto en el generador como en el discriminador.



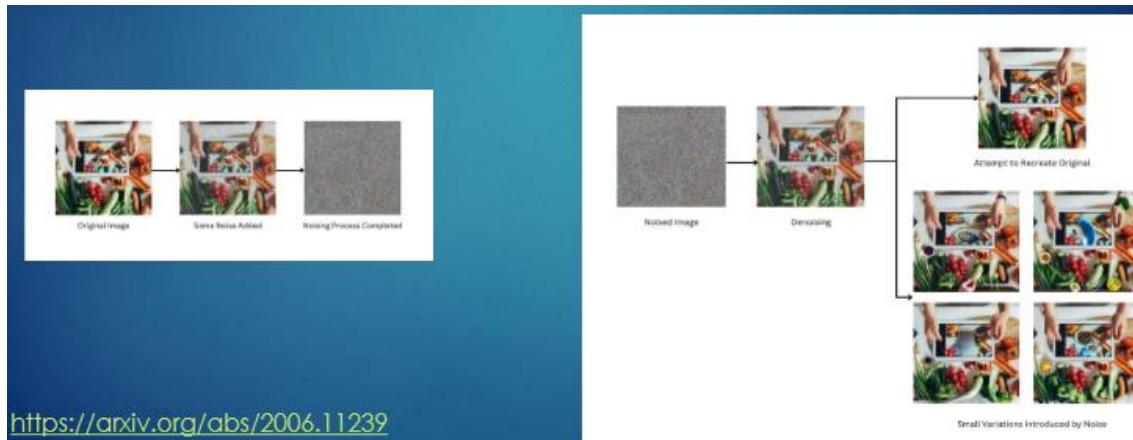
Se puede incluir mucha más información para condicionar la generación.



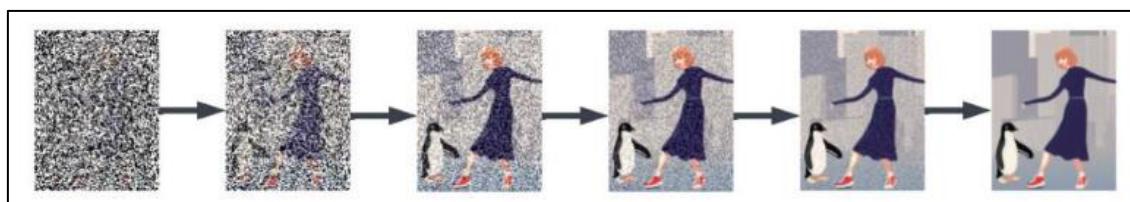
## Modelos de difusión

Ejemplos: Dall-E, Midjourney y Stable Diffusion

- La idea es añadir ruido a una imagen de a pasos, hasta que sea simplemente ruido gaussiano
- Luego se espera que el modelo aprenda a reconstruir la imagen, también paso a paso



- Agregarle ruido a una imagen es una tarea trivial
- Deshacer el ruido para generar una imagen nítida es el verdadero desafío



## Agregando ruido (difusión hacia adelante)

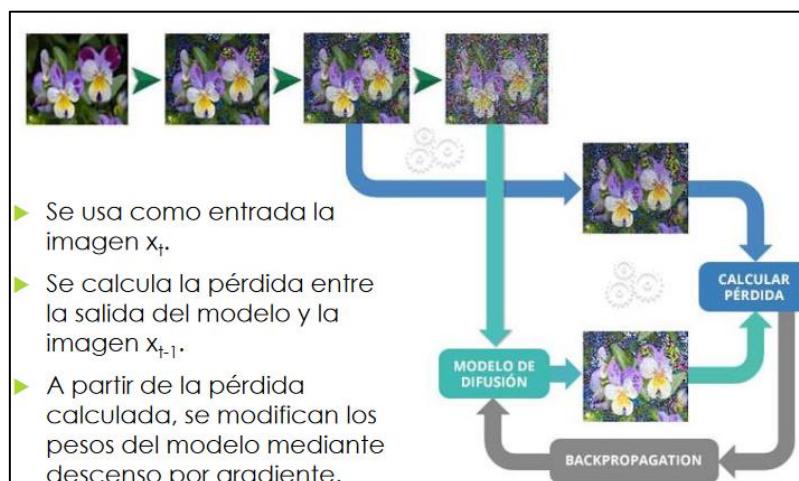
- El ruido se aplica siguiendo una Cadena de Markov.
  - o Se va aplicando ruido en sucesivos momentos (formando una cadena)
- La adición de ruido a la imagen en un determinado paso se realiza sobre la imagen del paso anterior.
- Por lo general se agrega ruido Gaussiano.
- El objetivo es generar las muestras de entrenamiento para el modelo de difusión.
- La Cadena de Markov suele ser de una longitud del orden de unas 1.000 adiciones de ruido.

- El ruido que se va aplicando en cada paso no siempre es el mismo.
  - o Se va regulando con un planificador que va cambiando poco a poco las formas de la distribución gaussiana
- Finalmente después de la última iteración se pierde totalmente la información de la imagen original.
  - o Pero con miles de modificaciones “pequeñas”.
- Luego el objetivo será “predecir” cuánto ruido se ha utilizado en cada paso, para poder ir eliminándolo y de esa manera llegar a la imagen original

## Reconstruyendo la imagen (difusión hacia atrás)

- Calcular la distribución inversa en su totalidad no es posible, pero al tratarse de una cadena de Markov es posible calcular una aproximación  $p(x_{t-1}|x_t)$ 
  - o es una manera de decir “calcular una imagen ligeramente menos ruidosa que la anterior”.
- El objetivo del entrenamiento de un modelo de difusión es calcular una distribución  $p(x_{t-1}|x_t)$  lo más cercana posible a la distribución real  $q(x_{t-1}|x_t)$

## Entrenamiento

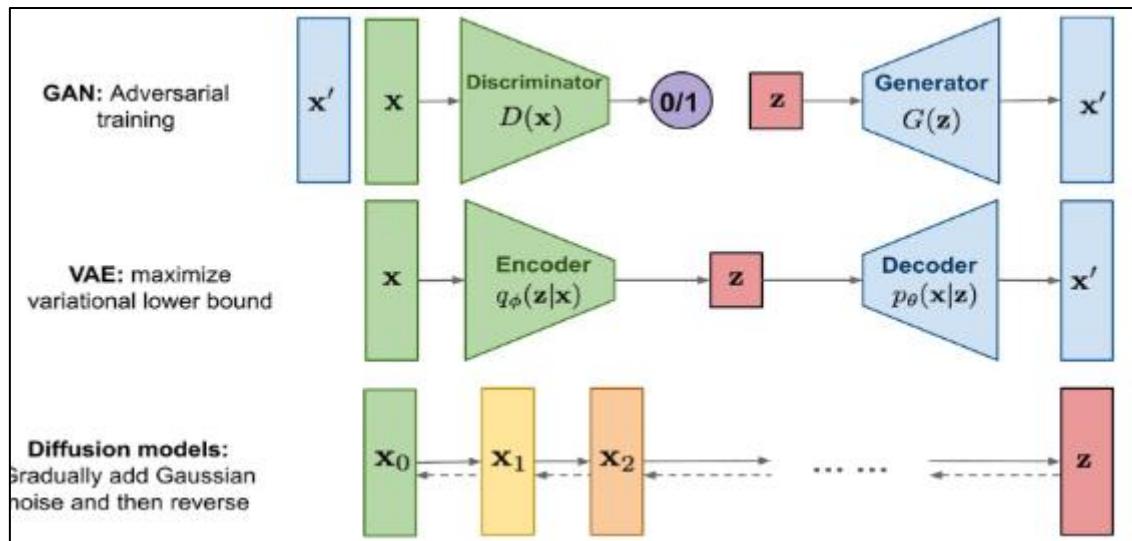


## Arquitectura

- Los modelos de difusión son típicamente UNets
- Tienen una primera etapa de contracción donde achican la imagen a un vector más pequeño, principalmente con capas convolucionales, y una segunda etapa donde a partir del vector generan una imagen del tamaño original.

## Ventajas

- Son mucho más estables en su entrenamiento que las GANs y no presentan los problemas ya vistos.
- Si no nos interesa trabajar con el espacio latente, son más sencillas de entrenar que los Autoencoders



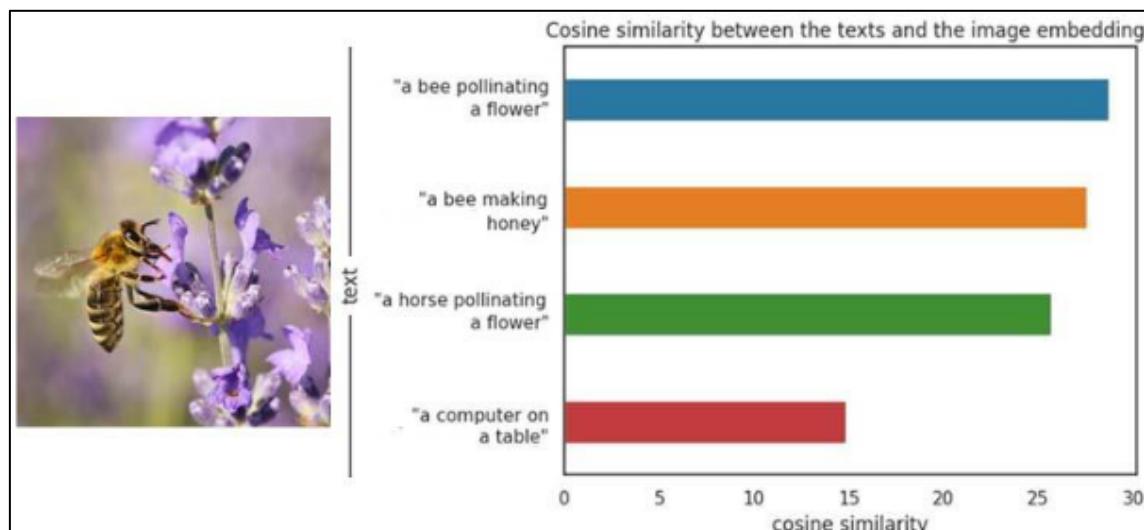
## Función de error

- Como función de error se suele utilizar la Divergencia Kullback-Leibler (KL).
- Interpreta las imágenes como una distribución de probabilidad y calcula la diferencia entre éstas.
- Se calcula este error para cada paso de la aplicación del modelo en función de los anteriores.

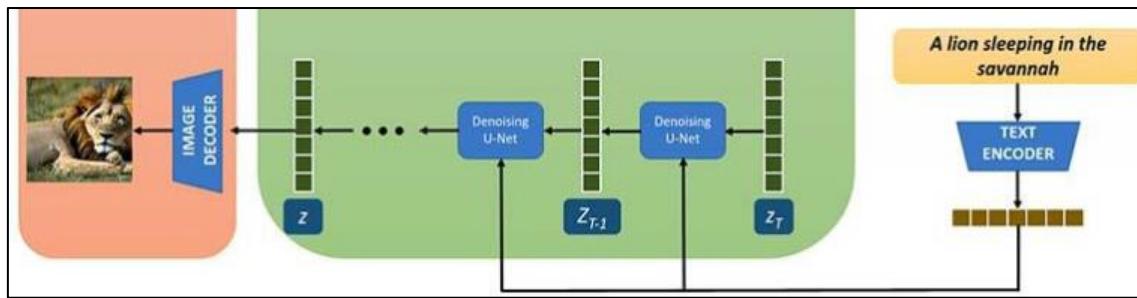
## Difusión guiada por texto

El objetivo es codificar texto e imágenes en un mismo espacio común.

- El entrenamiento se realiza a partir de imágenes con sus correspondientes descripciones textuales, con el fin de que el vector que representa a una imagen y el vector que representa a un texto sean más cercanos entre sí cuanto más relacionados estén dicha imagen y dicho texto.



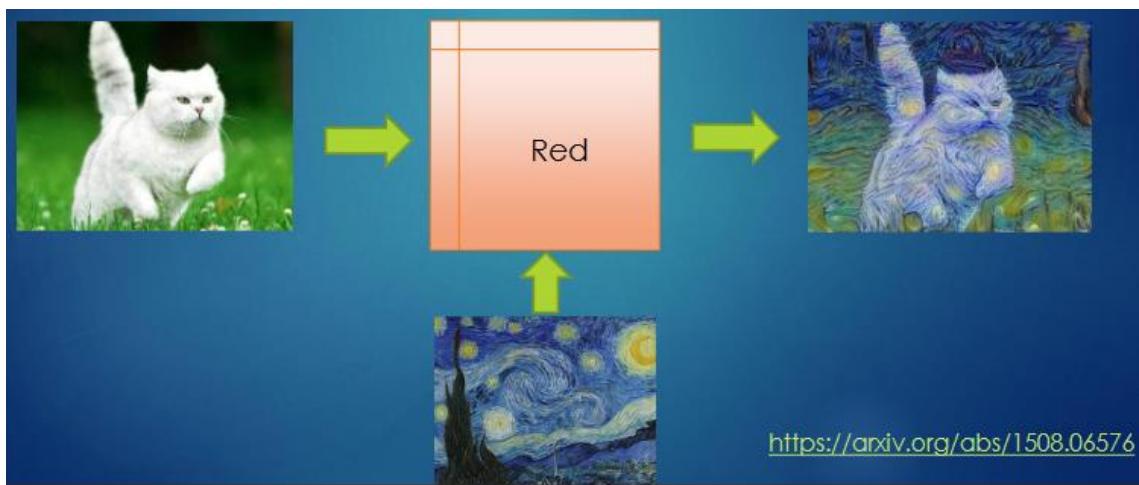
- El texto se codifica a su vector latente
- El mismo vector es utilizado en los T pasos de la reconstrucción de la imagen



## Neuronal Transfer Style (Transferencia de estilo)

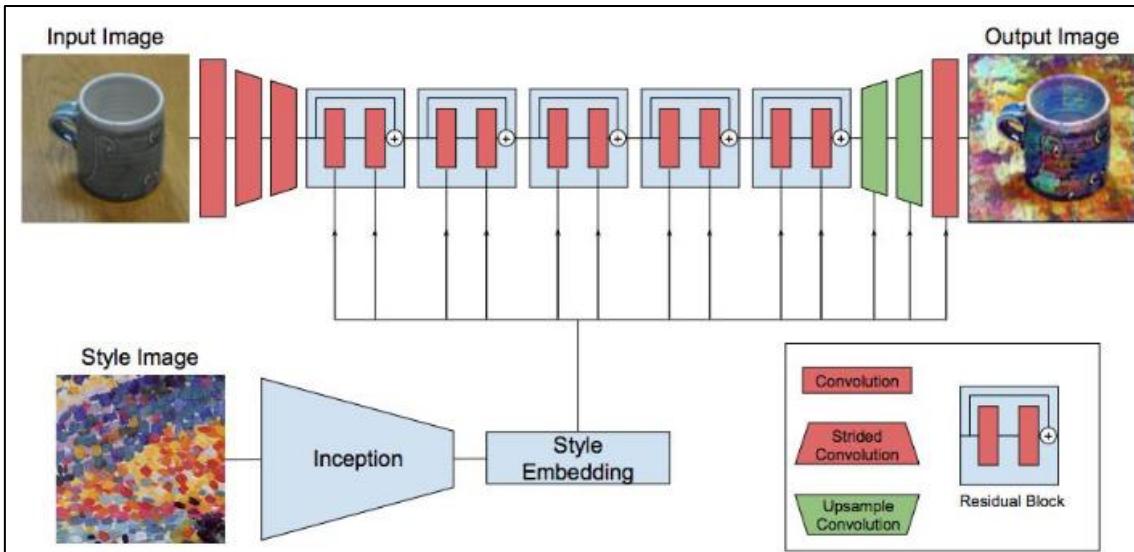


- La transferencia de estilo es una técnica que permite generar una imagen nueva desde dos imágenes existentes.
  - o El resultado será “el contenido” de una de las imágenes, pero con el estilo de la otra.
- Con el mismo concepto se pueden entrenar redes que aprendan “el estilo” de un conjunto de imágenes.



## Arquitectura

(En principio) cualquier red neuronal convolucional sirve para realizar transfer style



## Funcionamiento

- Solo se trabajan con tres imágenes
    - o La imagen con el contenido
    - o La imagen con el estilo
    - o La imagen resultante
      - Inicialmente esta imagen puede ser random o idéntica a la imagen-contenido original.
  - Los pesos de la red permanecen invariantes
  - La imagen resultante es la que se modifica para minimizar el error.

## Error a minimizar

Recordamos. Si tengo una red con una sola neurona y tres muestras de 1D, el error es:

$$\text{ECM} = (y_1 - x_1 \cdot w + b) + (y_2 - x_2 \cdot w + b) + (y_3 - x_3 \cdot w + b)$$


La salida de la red no se usa para nada.

## Función de error

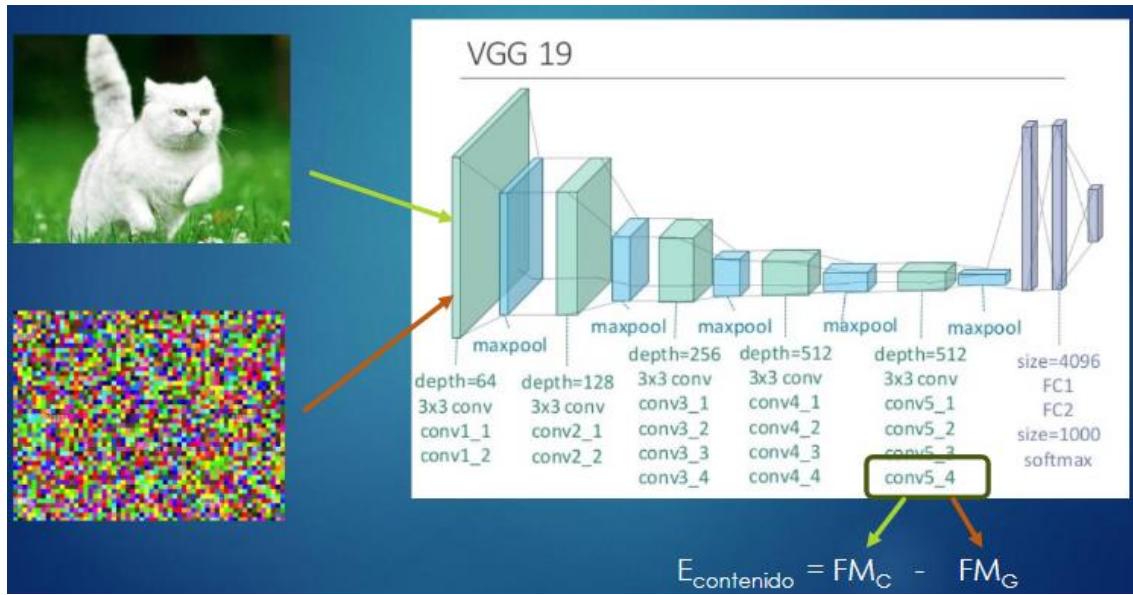
En este tipo de redes se deben optimizar dos aspectos, por lo tanto minimizar dos errores:

- El error de contenido de una imagen
  - El error de estilo de la otra

$$E(G) = \alpha E_{contenido}(C, G) + \beta E_{estilo}(S, G)$$

## Error de contenido

La premisa es que imágenes parecidas suelen tener activaciones parecidas en las capas profundas.

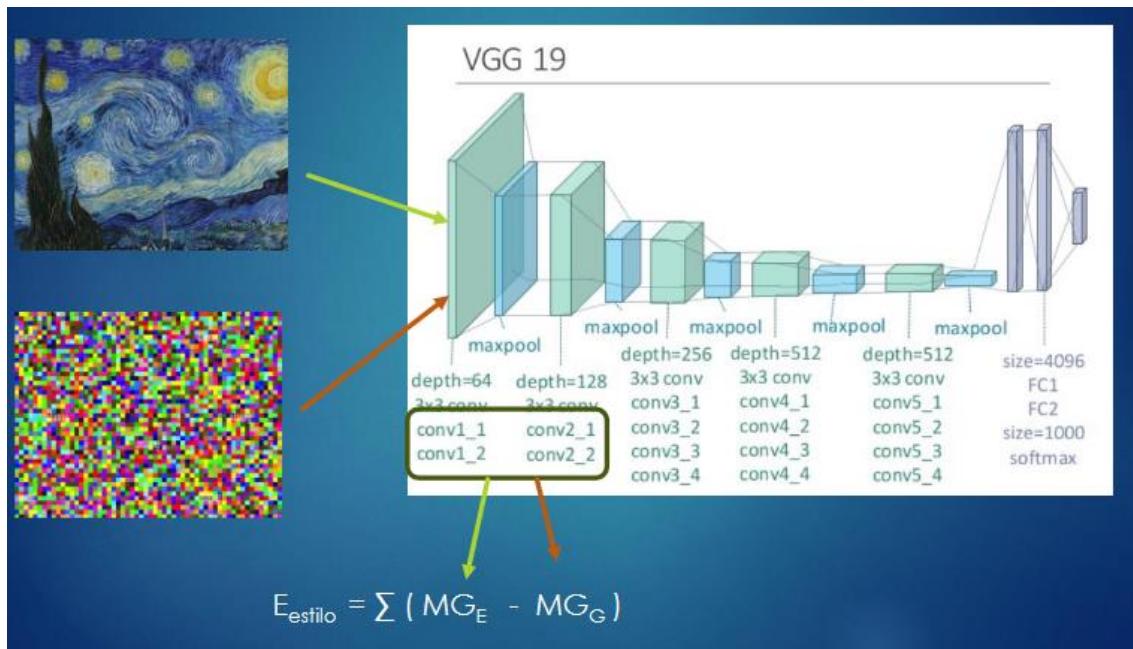
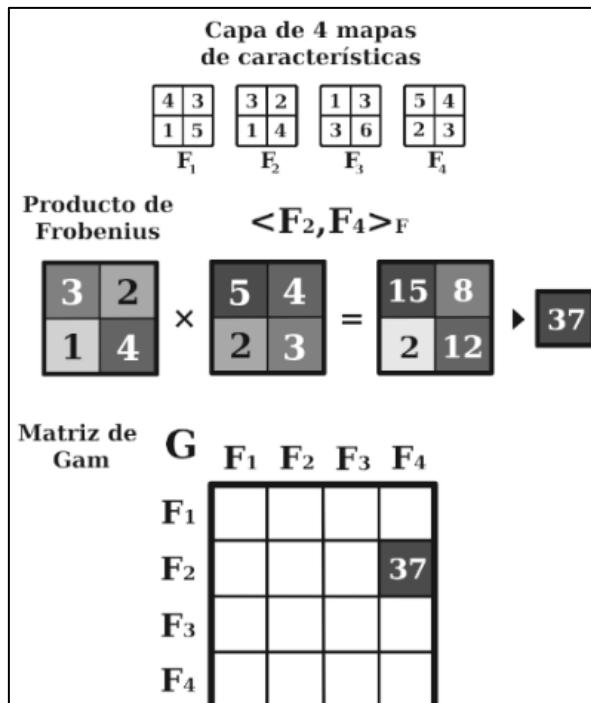


## Error de estilo

- El “estilo” de una imagen se determina por las pautas de coactivación existentes entre los pares de mapas de una misma capa.
  - o Una pauta de coactivación es la similitud (o disimilitud) de las distribuciones de valores de los feature maps de una capa.
- Para el cálculo del error de estilo se calculan las llamadas matrices Gram.
  - o A cada capa de la red le corresponde una matriz de Gram. Cada elemento de esta matriz es igual al producto de Frobenius de dos feature maps.

$$G_{i,j} = \langle F_i, F_j \rangle_F = \sum_{n=1}^{n=x} \sum_{n=1}^{n=y} (n_{x,y}, i^{n_{x,y,j}})$$

## Matrices gran



## Algoritmo

$G$  = Imagen generada al azar

$C$  = Imagen contenido

$E$  = Imagen estilo

Repetir hasta que el error no disminuya significativamente

- Obtener features maps de ( $G, C, E$ )
- Calcular gradientes

- Modificar G

## Redes recurrentes

Una **serie temporal** es un flujo de datos (finito o infinito) de una o más variables independientes, donde los datos de una misma variable son dependientes entre sí en el tiempo.

- Índices económicos
- Compras en tiendas
- Trayectorias
- Conversaciones
- Videos

El algoritmo de backpropagation sólo puede aprender relaciones estáticas. Una entrada X se asocia a una salida Y, es decir, se asocian patrones espaciales independientes del tiempo:

Las BPN se pueden utilizar para realizar predicciones no lineales de series de tiempo estacionaria.

- Una serie de tiempo es estacionaria cuando su “estadística” no cambia con el tiempo. En este caso X se puede definir como:  $X = [x(n-1), x(n-2), \dots, x(n-p)]$  donde p es el orden de predicción, y la salida de la red es un escalar  $y(n)$  que se produce como respuesta a la entrada X, representando la predicción del siguiente valor de X.

Las BPN siempre representan un modelo estático

- ¿Es posible representar el tiempo y proveer de propiedades dinámicas a la red?
- Para que una red sea dinámica, debe poseer "memoria".
- Existen dos maneras básicas de agregar "memoria" a una red neuronal
  - con retraso
  - con recurrencia

### Redes con retraso (variante que no se usa mucho)

Las redes neuronales con retraso incluyen un mecanismo de memoria introduciendo retrasos de tiempo en la estructura sináptica de la red y ajustando sus valores durante el entrenamiento.

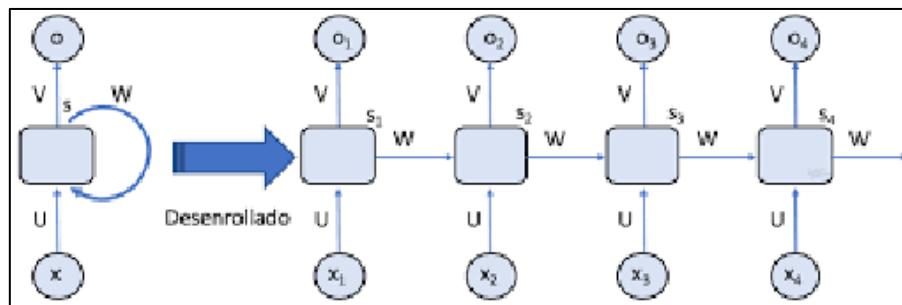
- Un ejemplo de este tipo de redes es la red "Time Delay Neural Network" (TDNN) descrita por Lang y Hinton en 1988 y por Waibel en 1989.
  - Es una red hacia delante de varios niveles cuyas neuronas ocultas y de salida se repiten a través del tiempo.

En las redes recurrentes hay dos metodologías básicas de entrenamiento:

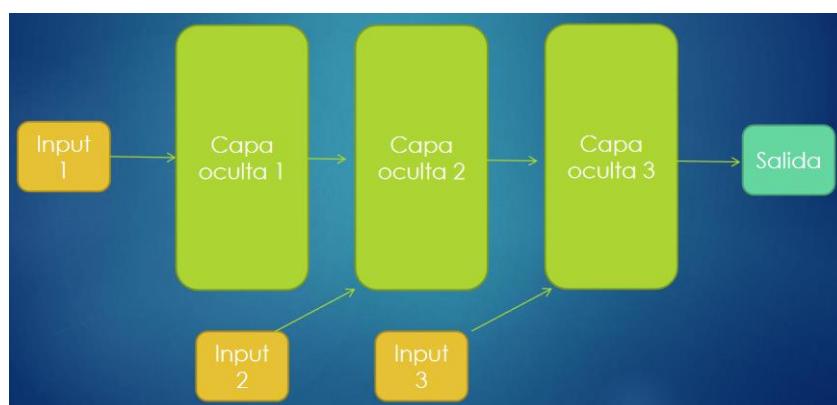
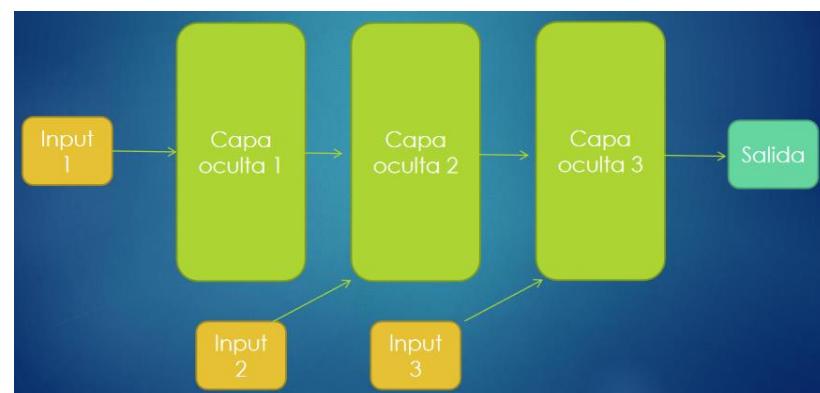
- Retropropagación a través del tiempo. Creada originalmente en la tesis de P. Werbos (1974), (1990). Redescubierta independientemente por Rumelhart et al. (1986) y una variación propuesta por Williams y Peng (1990).

- Aprendizaje Recurrente al Tiempo Real (Real Time Recurrent Learning). Describo por Williams y Zipsen (1989), los orígenes del algoritmo fueron dados por McBride y Nardendra (1965)

Las redes neuronales recurrentes constituyen una herramienta muy apropiada para modelar series temporales. Se trata de un tipo de redes con una arquitectura que implementa una cierta memoria y, por lo tanto, un sentido temporal. Esto se consigue implementando algunas neuronas que reciben como entrada la salida de una de las capas posteriores e inyectan su salida en una de las capas anterior a ella.

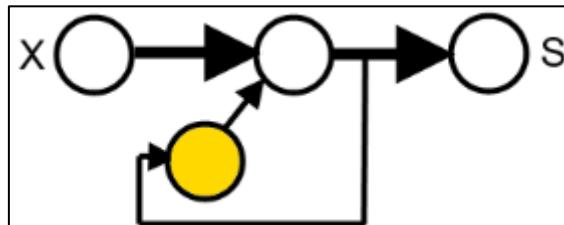


El objetivo es encontrar un mecanismo que aprenda de lo nuevo y de lo viejo al mismo tiempo, sin tener que definir un número explícito de capas.

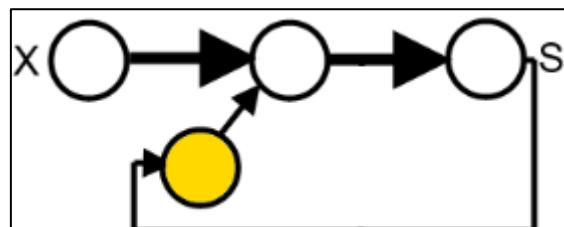


Las arquitecturas más básicas de este tipo de redes neuronales son las conocidas como las redes de Elman y las redes de Jordan

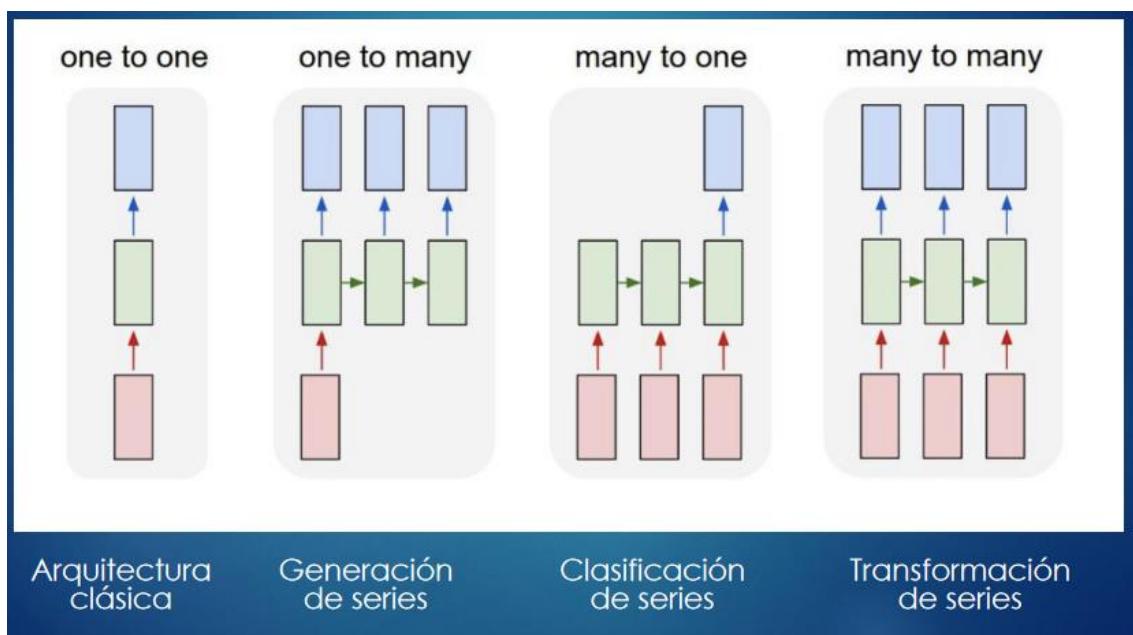
En las redes de **Elman**, las entradas se toman desde las salidas de las neuronas de una de las capas ocultas, y sus salidas se conectan de nuevo en las entradas de esta misma capa, proporcionando una especie de memoria sobre el estado anterior de dicha capa.



En las redes de **Jordan**, la diferencia está en que la entrada de las neuronas de la capa de contexto se toma desde la salida de la red.



## Arquitecturas de redes neuronales



Una red recurrente se puede ver como una máquina de estados. Un dato llega a la red, la cual se encuentra en un estado t.

Después de analizar un dato, la red cambia a un estado t+1

La fórmula para el estado actual de una RNN es:

$$h(t) = f(h(t-1), x(t))$$

El estado h para una neurona de la capa oculta se calcula:

$$h(t) = \text{sig}(W_{hh} * h(t-1) + W_{xh} * X(t))$$

Esta fórmula solo toma el último estado como memoria. Para incrementar la memoria se deben agregar más estados a la red.

$$h(t) = \text{sig}(\sum [W_{hh} * h(t-p)] + W_{xh} * X(t))$$

Luego, la salida de una neurona de la capa oculta se calcula como:

$$y(t) = W_{hy} * h(t)$$

## Entrenamiento (Backpropagation Through Time)

El backpropagation estándar no puede usarse con las RNN, dada su característica cíclica.

El entrenamiento de una RNN se lo conoce como **Backpropagation Through Time (BPTT)**.

Se busca minimizar el error obtenido en un período de tiempo entre la salida obtenida y la esperada.

$$E = \int_{t_0}^{t_n} (y(t) - d(t)) dt$$

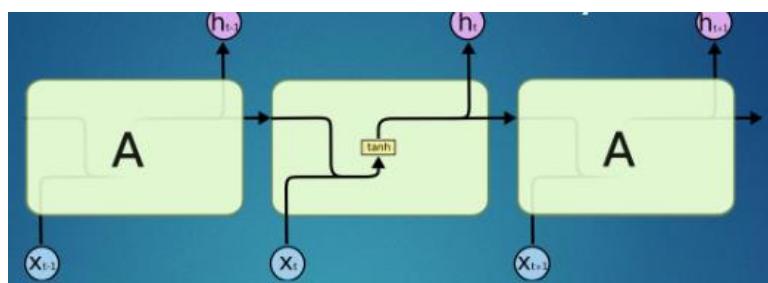
Donde  $y(t)$  es la salida obtenida y  $d(t)$  es la salida esperada

Se busca la minimización de  $E$ .

## Long-Short Term memory

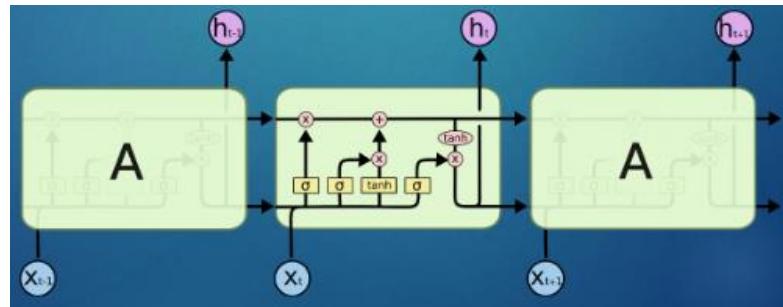
Las redes Long-Short Term Memory (LSTM) son una extensión de las RNN.

El objetivo de este tipo de redes es poseer una gran cantidad de memoria, la cual puede administrar agregando, actualizando y eliminando información, permitiendo así que este tipo de redes recuerden los diferentes datos que entraron a lo largo del tiempo.



Este tipo de redes están compuestas por neuronas las cuales almacenan los datos de entrada por un período de tiempo y además poseen "compuertas" (gated cell) que se encargan de decidir si se debe almacenar o eliminar datos de entrada (según su estado interno: abierta o cerrada)

La decisión de almacenar o no la información está basada en los pesos de una neurona, esto implica que de alguna forma este tipo de redes aprende a distinguir la importancia de un dato.



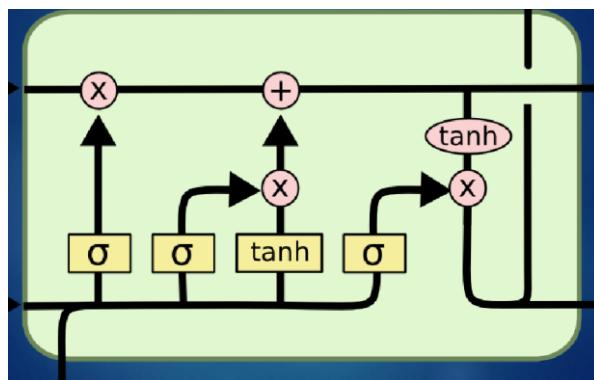
## Compuertas

Una LSTM está formado de compuertas, las cuales son neuronas que pesan la entrada y producen una salida acotada usando una función de activación sigmoidal.

Se ha demostrado que el uso de estas compuertas soluciona el problema de los "vanishing gradients" ya que se consigue que el gradiente tenga cierta "inclinación" logrando que el entrenamiento sea rápido así como también un resultado aceptable.

Existen tres tipos de compuertas:

- De entrada: deciden si almacenar o no al dato
- De olvido: eliminar un dato (porque ya no es importante)
- De salida: permitir que un dato sea usado en el siguiente paso de la RNN



Los sigmas son las compuertas de entrada, olvido y salida (de izquierda a derecha).

Abajo: dato nuevo

Abajo a la izquierda: salida anterior

Arriba a la izquierda: estado anterior

Salida a la derecha abajo: va a la siguiente iteración que entra a la misma neurona

Salida a la derecha arriba: va a la siguiente neurona

Salida arriba: predicción

## Usos de las LSTM

Las LSTM se han utilizado en diferentes campos:

- Predicción en la escritura
- Detección de anomalías en series temporales

- Predicción en los mercados
- Reconocimiento del habla
- Predicción de trayectorias

## Few-Shot Learning

Uno de los principales problemas que tienen las redes neuronales (como así también otros algoritmos de machine learning) es que necesitan una “considerable” cantidad de datos etiquetados para poder armar un modelo.

Problemas con modelos en producción:

- Aparecen nuevos datos (deriva del concepto)
- Aparecen problemas parecidos
- Aparecen nuevas clases de datos

Estos son problema de Transfer learning Data augmentation

¿Cómo evitamos tener que entrenar una y otra vez con nuevos datos?

- Usar la técnica conocida como few-shot learning (aprendizaje con pocas muestras).

**Few-shot learning** permite usar modelos pre-entrenados para “extenderlos” y que puedan aprender de nuevos ejemplos, sin necesidad de reentrenar el modelo.

Few-shot learning es un conjunto de técnicas que consiguen modelos efectivos a partir de pocos ejemplos etiquetados.

Mayormente se lo utiliza en tareas de clasificación, aunque también se lo puede usar en regresión o en aprendizaje por refuerzo.

Few-shot learning pretende emular la capacidad humana de aprender de pocos casos.

- Aprendizaje por transferencia
  - o Modificar la arquitectura, en especial las últimas capas.
  - o La transferencia funciona mejor si el modelo pre-entrenado es del mismo dominio que el nuevo problema.
- Generación de datos
  - o Redes generativas, autoencoders variacionales.
  - o Aumentación de datos.
- Meta aprendizaje
  - o Son conjuntos de tareas bien específicas que se “aprenden” por separado en diferentes tareas (también llamados episodios).
  - o Luego de un número considerable de tareas, el modelo acumula cierto conocimiento, el cual le permite aprender nuevas tareas de manera más eficiente.
    - Aprende a aprender.

## Meta aprendizaje

Un algoritmo se dice que “aprende” si su performance ante nuevos problemas mejora con la experiencia ganada previamente.

- “Aprende” a “aprender”

La idea es que si un algoritmo fue entrenado en varias tareas, puede ser que el conocimiento adquirido le sirva para resolver una nueva tarea nunca antes vista

- Basado en el hecho que a los humanos le resulta fácil aprender “nuevas” clases de datos usando su conocimiento previo.

Por lo general el meta aprendizaje consiste de un número finito de episodios.

## Clasificación N-way-K-shot

Esquema N-way K-shot: N nuevas clases de K muestras cada una.

- Por lo general K es chico (diez o menos).
- Con  $k=1$  se habla de one-shot learning.
- Con  $k=0$  se habla de zero-shot learning.

El modelo es entrenado en varios episodios de entrenamiento

Un episodio consiste en uno o más tareas de entrenamiento

El modelo se evalúa con tareas de prueba.

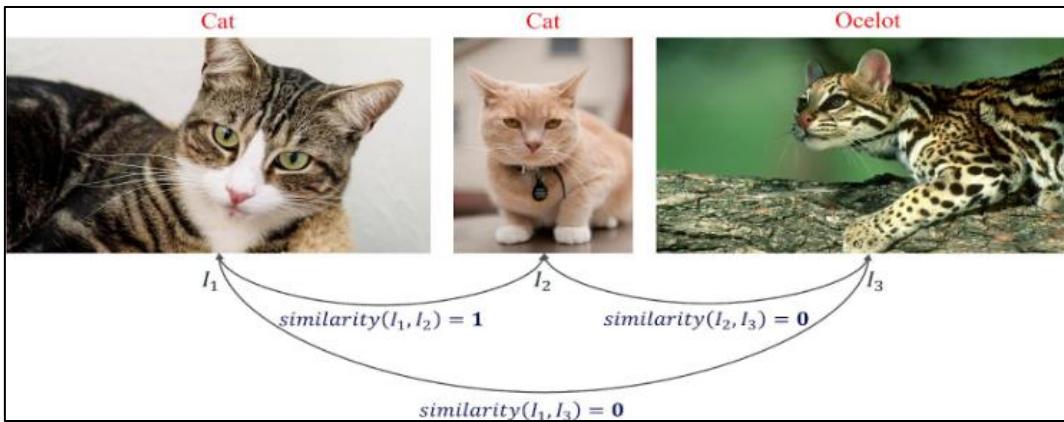
Cada tarea está compuesta por:

- Un conjunto de soporte: son “pocas” K nuevas muestras para cada una de las N clases.
- Conjunto de consulta: Consiste en nuevas muestras para las N clases.

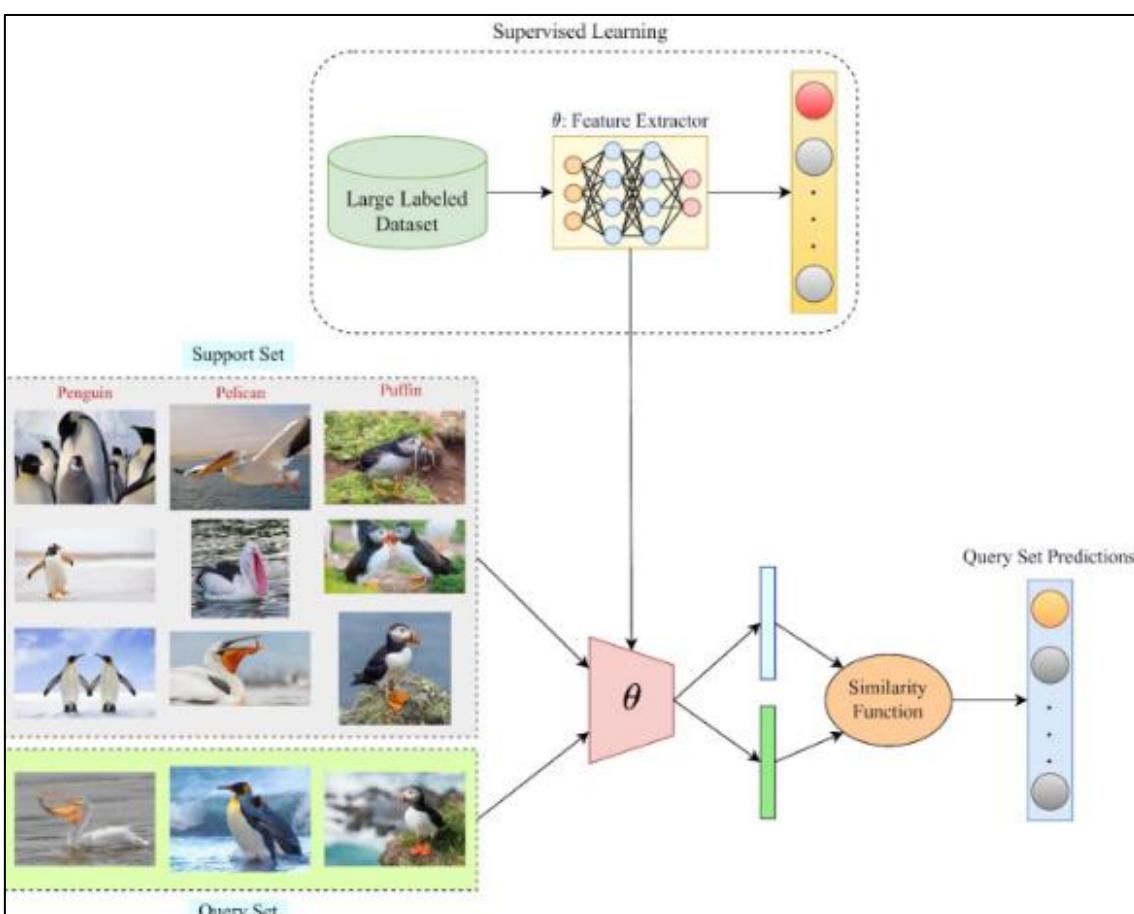
► Por lo general cada tarea tiene muestras de clases diferentes.



El objetivo de un modelo en few-shot learning no es dar una clasificación como respuesta. Sino la de aprender una función de similitud. Por ejemplo, si muestro dos fotos de gatos dan similitud cerca de 1, pero si muestro un gato y un ocelote muestro una probabilidad de similitud menor.

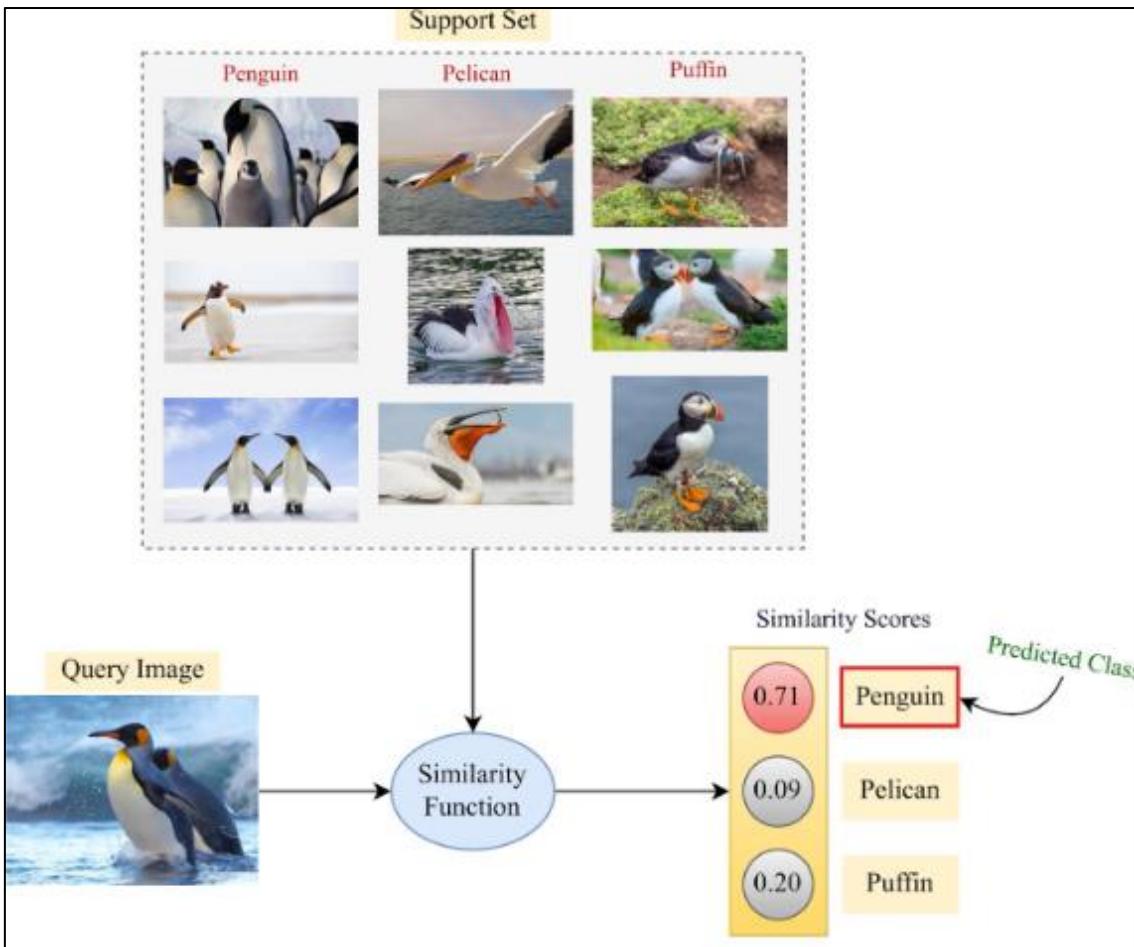


Podemos usar los modelos de redes neuronales pre-entrenados para usarlos como base y a partir de ellos construir modelos de “similitud”.



Se usa un modelo pre-entrenado y el dataset de soporte para conseguir un modelo que actue como función de similitud.

Luego se usa este modelo-función para inferir nuevas muestras



## Redes siamesas

Few-Shot Learning necesita una función de similitud. Pero no necesariamente tiene que ser una función.

- Puede ser una red neuronal

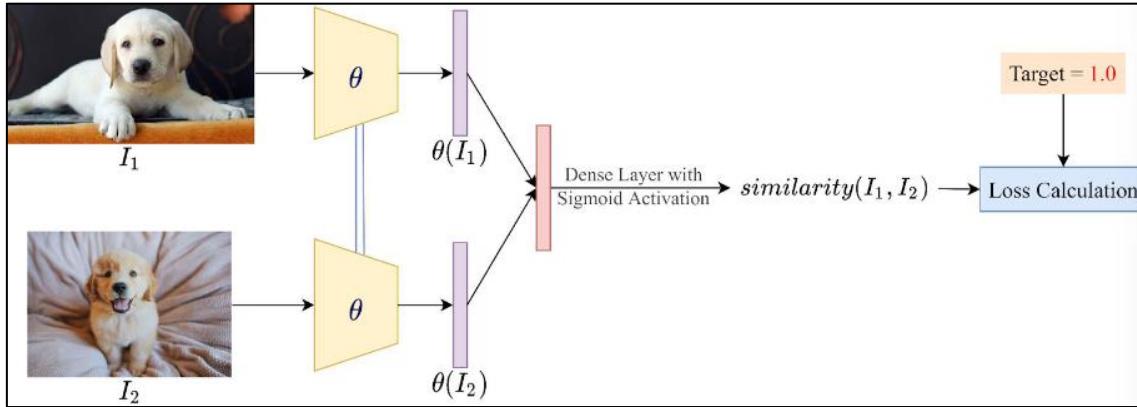
Hay dos enfoques para el entrenamiento de redes siamesas.

- Similitud entre pares (pairwise similarity)
- Pérdida de tripletes (triplet loss)

### Similitud entre pares

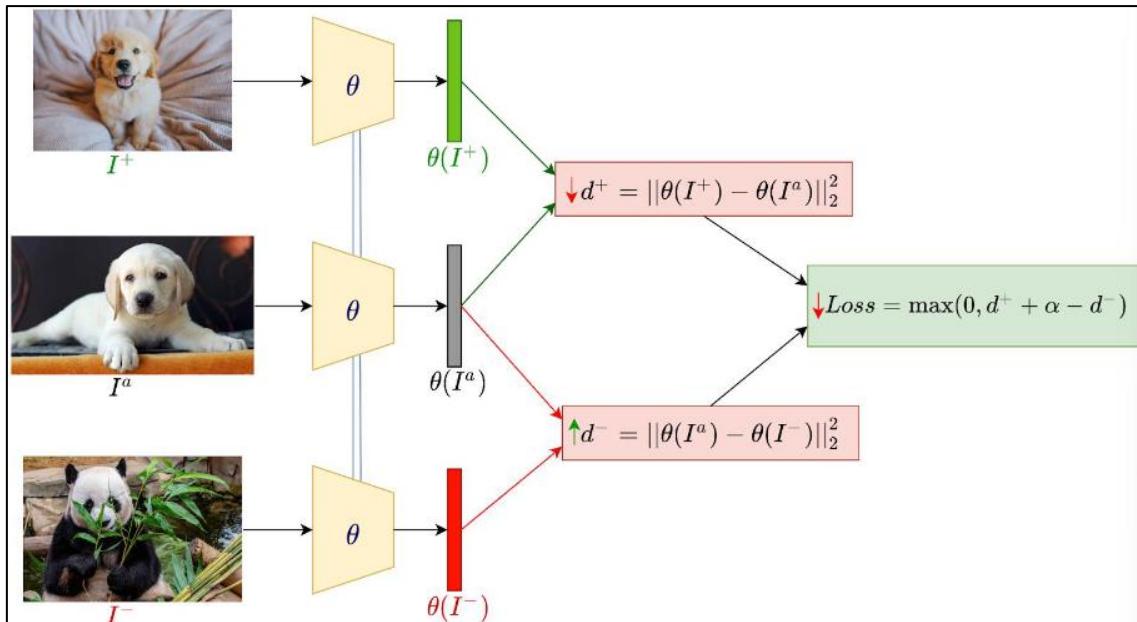
- La red recibe dos entradas con sus correspondientes etiquetas
- Se selecciona una imagen aleatoria del dataset de soporte: ( $I_1, C_1$ )
- Luego se elige otra imagen ( $I_2, C_2$ ).
  - o Si  $C_1 = C_2$  se asigna un 1 como target
  - o en caso contrario se asigna un 0.
- La red deberá aprender a asociar que las dos imágenes  $I_1$  e  $I_2$  son similares (en relación a su clase) o si son distintas.
- Ambas imágenes pasan por la misma red pre-entrenada.
  - o Se busca usar un feature map “descriptivo” que extraiga una “representación” (embedding).

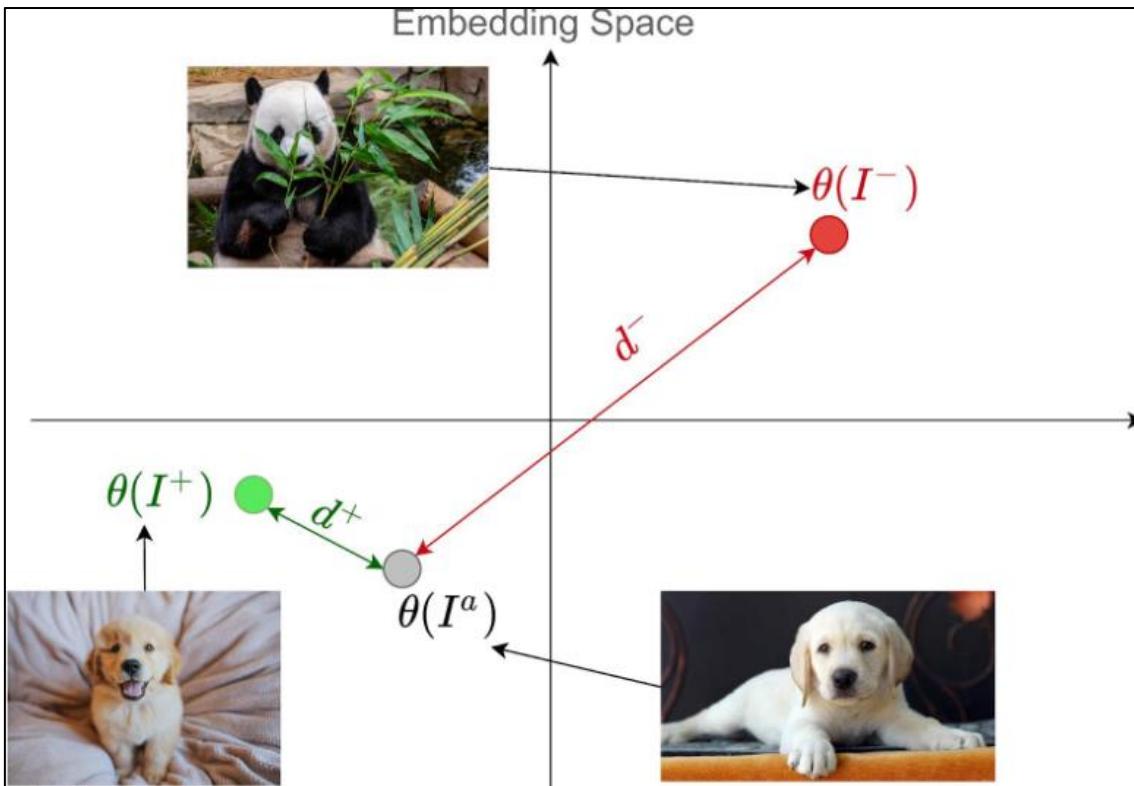
- Las representaciones de ambas imágenes se concatenan para que la(s) nueva(s) capa(s) aprenda(n) la función de similitud.



## Pérdida de tripleta

- Es una extensión del método de similitud entre pares.
- Se elige una muestra del dataset de soporte la cual llamaremos “ancla”, la cual pertenece a una clase C.
- Luego se elige otra imagen de la clase C (llamada muestra positiva) y una tercer imagen de otra clase distinta de C (llamada muestra negativa).
- Las tres imágenes pasan por el mismo modelo pre-entrenado.
  - o Se calcula la distancia normalizada L2 entre la muestra “ancla” y la muestra positiva:  $d^+$ .
  - o Se calcula la distancia normalizada L2 entre la muestra “ancla” y la muestra negativa:  $d^-$ .
- La función de pérdida está definida con ambas distancias.
  - o Se debe minimizar  $d^+$  y maximizar  $d^-$

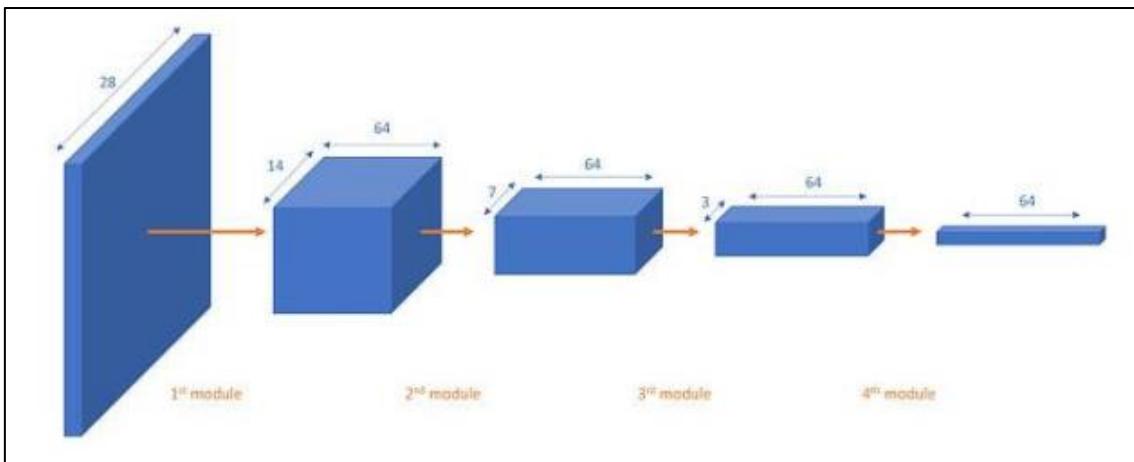




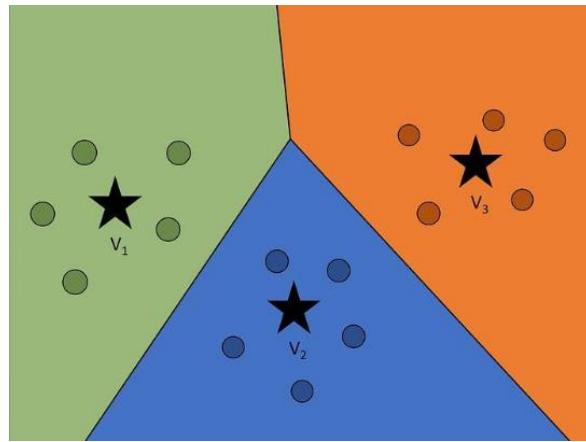
## Redes prototípicas

La idea principal es que existe los embeddings de muestras de la misma clase forman un cluster.

- La arquitectura está formada por bloques
  - o Cada bloque tiene capas convolucionales, capas batch normalization y capas max pooling.
- La salida es un vector de N dimensiones.



Con los embeddings de cada clase se busca un centroide, al mejor estilo k-means.



Para cada muestra se mide la distancia con cada centroide de cada clase. Luego se aplica una softmax para calcular la probabilidad de pertenecer a una clase.

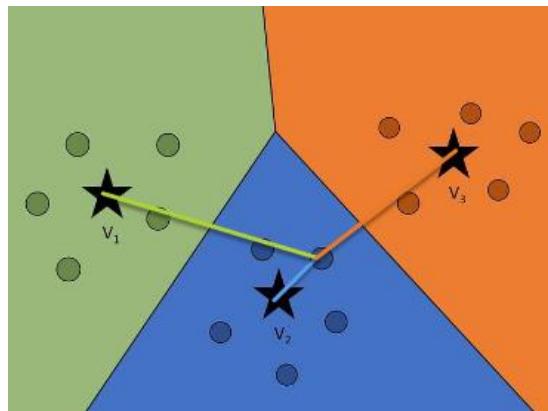
$$p_i = \text{softmax}(-d_i^{C1}, -d_i^{C2}, -d_i^{C3})$$

Entonces el error de una muestra es:

$$\text{error}_i = -\log(p_i)$$

La función de error a minimizar es:

$$E = \sum \text{error}_i$$



## Meta aprendizaje

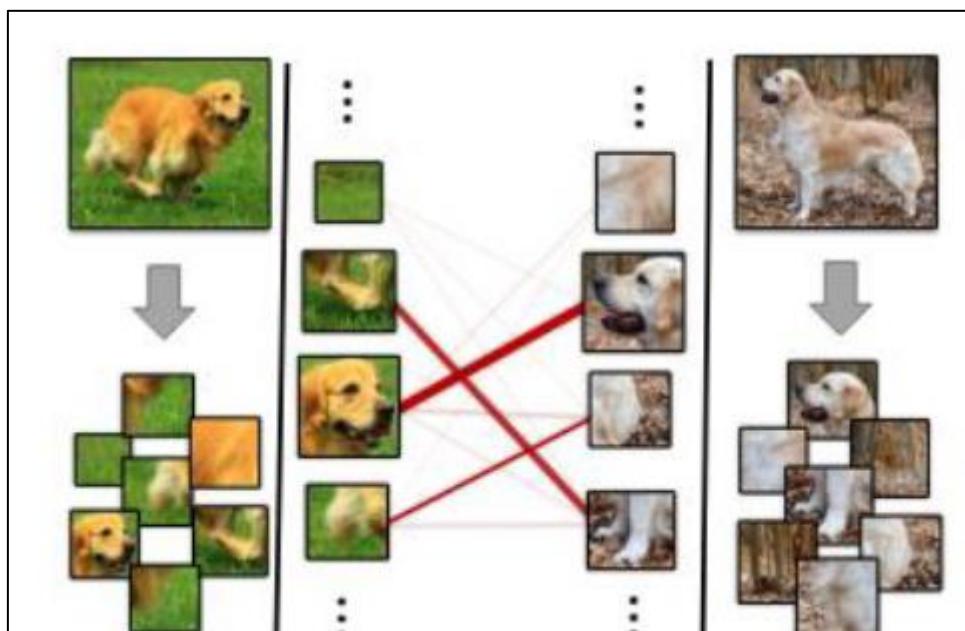
Metaaprendizaje de modelo agnóstico (MAML)

- Optimiza los hiperparámetros de un algoritmo
- El algoritmo del descenso del gradiente optimiza los parámetros de la red.
- Después de entrenar cada tarea se elige una muestra al azar y con esa muestra se aplica el descenso del gradiente para optimizar los parámetros iniciales.
- La ganancia se consigue luego de la optimización de varias tareas.
- El objetivo es encontrar los parámetros de modelo que son sensibles a los cambios en la tarea.
  
- Meta aprendizaje LTSM

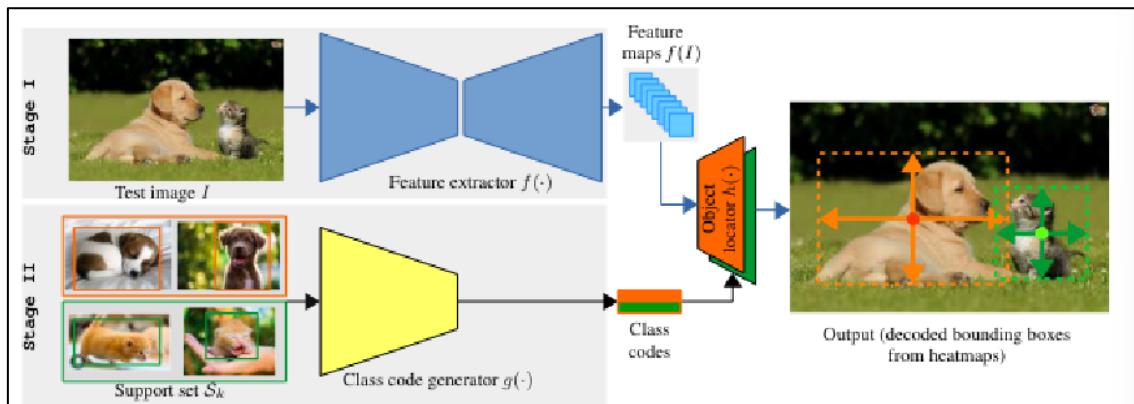
- Se usa para capturar el conocimiento a corto plazo de cada tarea de entrenamiento, así como el conocimiento a largo plazo común a cada tarea.
- Optimización de la incrustación latente (LEO)
  - Intenta aprender una distribución de los parámetros del modelo específicos de la tarea.
    - Funciona de manera similar a los autocodificadores variacionales (VAE)

## Usos del Few-shot learning

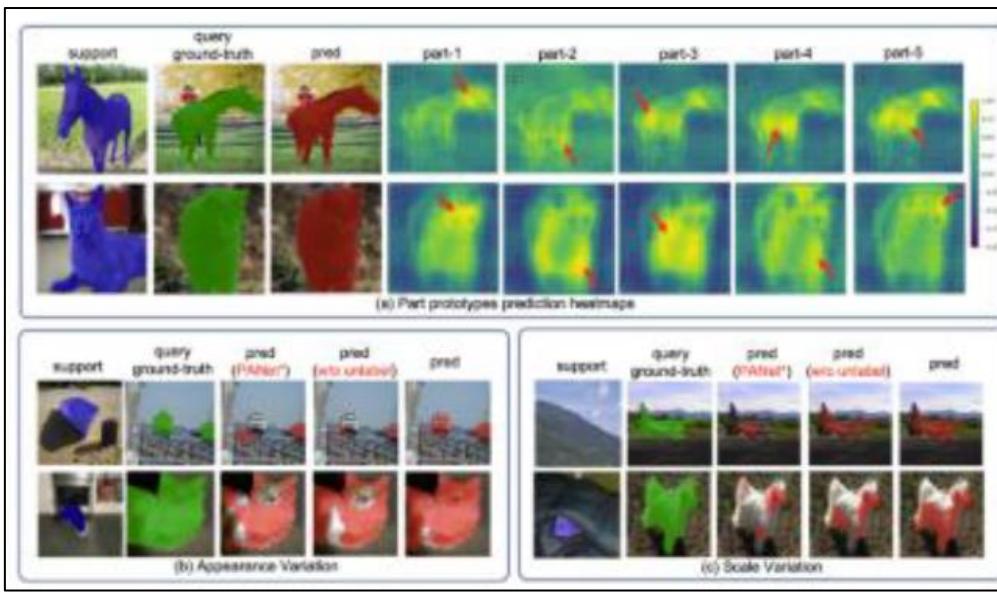
- Clasificación de imágenes
- Una forma de clasificar imágenes es a través de detectar los “bloques constructivos” de l objeto a clasificar. Si dos imágenes tienen los mismos bloques, entonces tienen el mismo objeto.



- Detección de objetos

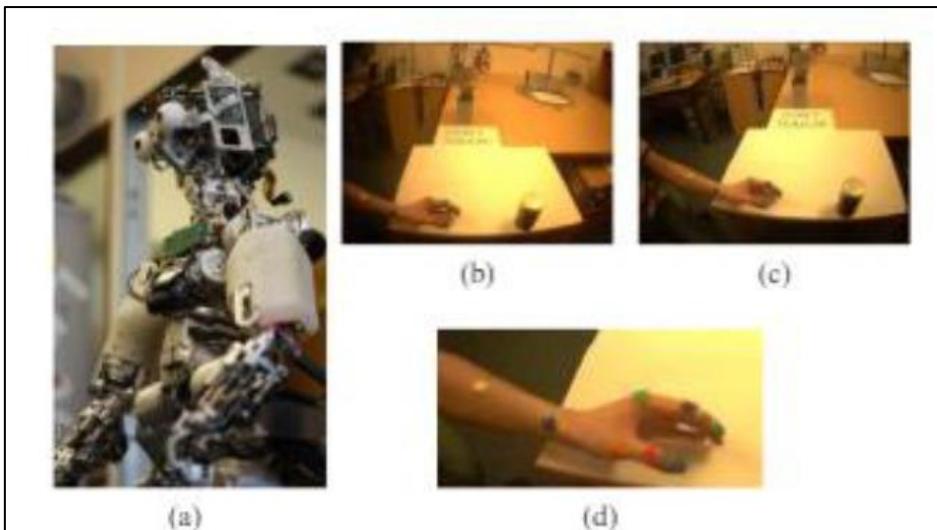


- Segmentación semántica



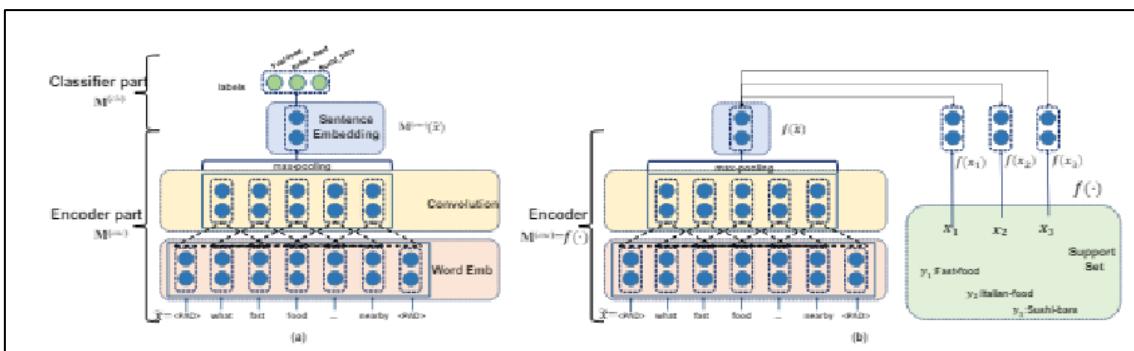
- Robótica

En el campo de la robótica resulta útil que los robots imiten ciertos movimientos humanos a partir de pocas observaciones

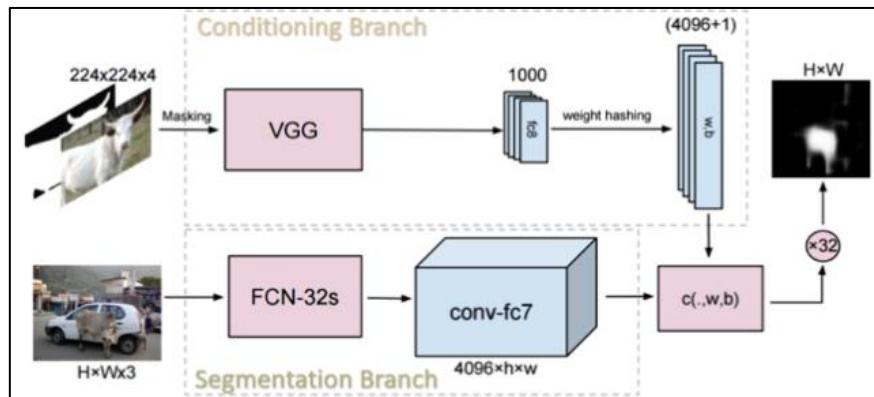


- Procesamiento del lenguaje natural

En problemas de clasificación de textos, few-shot learning resulta una técnica útil cuando hay pocas muestras.



- Un ejemplo de one-shot learning consiste en resolver un problema de segmentación semántica.
- Una entrada del modelo toma una imagen etiquetada (objeto que se desea encontrar) y produce un vector de parámetros ( $w, b$ ).
  - o Se computa en una sola pasada.
- La otra entrada toma la imagen a segmentar y el vector de parámetros y produce una imagen con la máscara.



## Detalles técnicos de los modelos

Hasta ahora solo nos preocupaba la performance del modelo

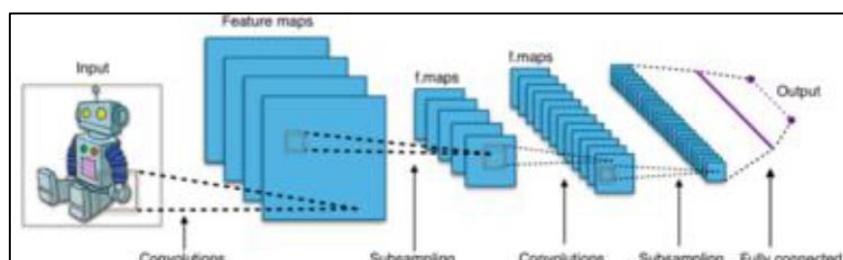
- Y eventualmente el tiempo de entrenamiento

Pero...

- ¿Cuánto pesa nuestro modelo?
- ¿En dónde lo podemos usar en producción?
  - o Servidores
  - o Aplicaciones de escritorio
  - o Aplicaciones para teléfonos móviles
  - o Robótica
    - Microcontroladores

Otros parámetros no estudiados hasta ahora es el tiempo de inferencia y el consumo energético.

- Tiempo de inferencia (o latencia) es el tiempo en el cual se presenta una muestra al modelo y este hace todos los cálculos necesarios para obtener la respuesta.
- Consumo es la energía necesaria para llevar a cabo todas las operaciones aritmético-lógicas que permiten hacer la inferencia (consumo de inferencia)



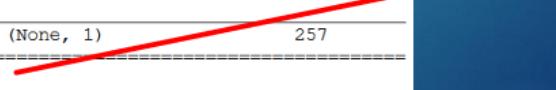
## Peso de un modelo

El peso de un modelo es lo que ocupa en memoria RAM para poder hacer inferencias.

- Parámetros del modelo
- Algoritmo de inferencia
- Muestra a presentar y resultado obtenido

model.summary()		
Layer (type)	Output Shape	Param #
vgg19 (Functional)	(None, 4, 4, 512)	20024384
flatten (Flatten)	(None, 8192)	0
dense_2 (Dense)	(None, 256)	2097408
dense_3 (Dense)	(None, 1)	257
Total params: 22,122,049		
Trainable params: 22,122,049		
Non-trainable params: 0		

22.122.049  
4  
88.488.196  
84.3 MB



- Dall-E-2 → 3.500.000.000 parámetros
- LLaMa-2 → 7.000.000.000, 13.000.000.000 y 70.000.000.000
- GPT-3 → 175.000.000.000 de parámetro

### ¿Podemos hacer adelgazar a un modelo?

El proceso que transforma un modelo en otro con las “mismas” capacidades de predicción pero que ocupe menos memoria se lo conoce como cuantificación de modelos.

La cuantificación de redes neuronales permite reducir el tamaño y la complejidad de los modelos.

- Al cuantificar una red neuronal, se puede representar cada peso y activación con menos bits, lo cual reduce significativamente el consumo de memoria y energía.
- La cuantificación de redes neuronales también permite acelerar el proceso de inferencia, ya que se requieren menos operaciones aritméticas para realizar los cálculos.
- Los dispositivos pueden procesar más información en menos tiempo, lo cual es crucial en aplicaciones en tiempo real.

## Cuantificación de redes neuronales

La cuantificación de redes neuronales es el proceso de representar los pesos y activaciones de una red neuronal con menos bits que en la representación original de punto flotante de 32 bits.

Esto implica asignar un rango de valores a cada peso y activación y redondearlos al valor más cercano dentro de ese rango.

- Con 32 bits → 4.294.967.296 posibles valores
- Con 16 bits → 65.536 posibles valores
- Con 8 bits → 256 posibles valores

Si nuestro modelo trabaja con pesos entre 0 y 1 entonces:

- Con 32 bits: resolución de  $1 / 4.294.967.296 = 0.00000000023$
- Con 16 bits: resolución de  $1 / 65.536 = 0.000015$
- Con 8 bits: resolución de  $1 / 256 = 0.0039$

Cuántos menos bits usemos para cada parámetro, menos posibles valores vamos a poder representar.

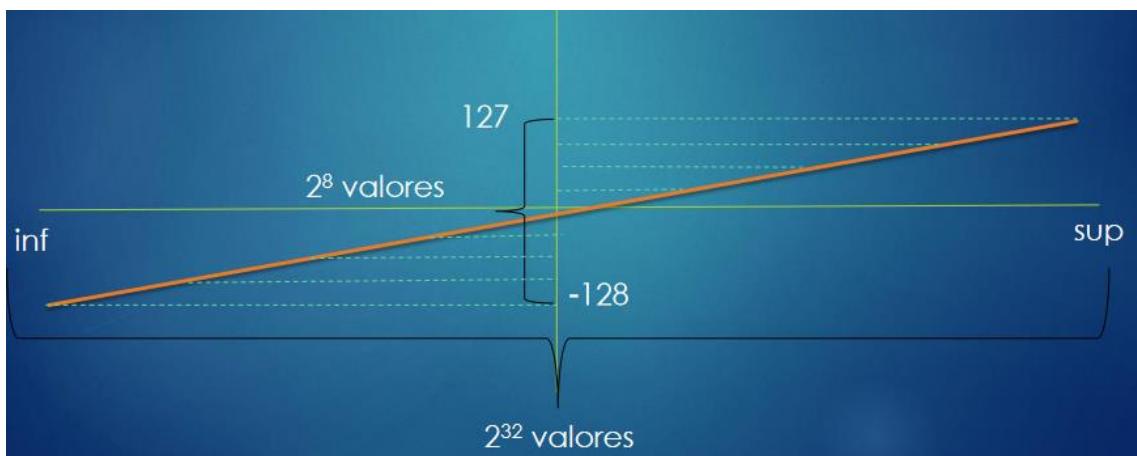
Para cuantificar N bits a Z bits se usa la ecuación:

$$y = mx + b$$

$$m = (2Z + 1)/(sup - inf)$$

$$b = -(ROUND(inf * m)) - 2(Z - 1)$$

Los valores de z se redondean al más cercano dentro de la precisión buscada



#### *Cuantificación durante el entrenamiento*

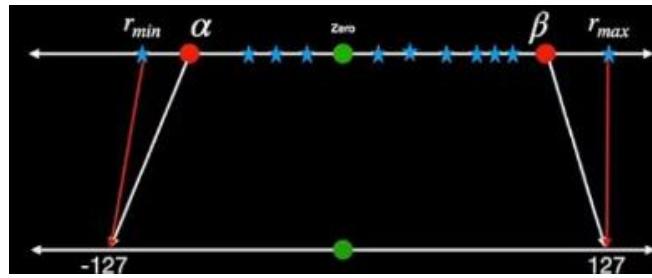
La cuantificación durante el entrenamiento es un enfoque que implica entrenar la red neuronal directamente con un número reducido de bits. Esto se logra mediante la cuantificación de los pesos y activaciones durante el proceso de entrenamiento.

La técnica conocida como "destilación del conocimiento" (knowledge distillation), consiste en entrenar una red neuronal de alta precisión (32 bits) y luego utilizarla como maestra para entrenar una red más pequeña con menos bits.

La técnica "poda de pesos" (weight pruning) implica eliminar los pesos de la red neuronal que tienen un impacto mínimo en el rendimiento. Esto reduce el número total de pesos en la red y facilita la implementación de cuantificación con pocos bits.

La cuantificación posterior al entrenamiento implica cuantificar una red neuronal previamente entrenada. O sea, una vez que ya se conocen todos los pesos de la red.

- Durante la cuantificación posterior al entrenamiento, se realiza un análisis de las estadísticas de los pesos y activaciones de la red para determinar los rangos de cuantificación óptimos.



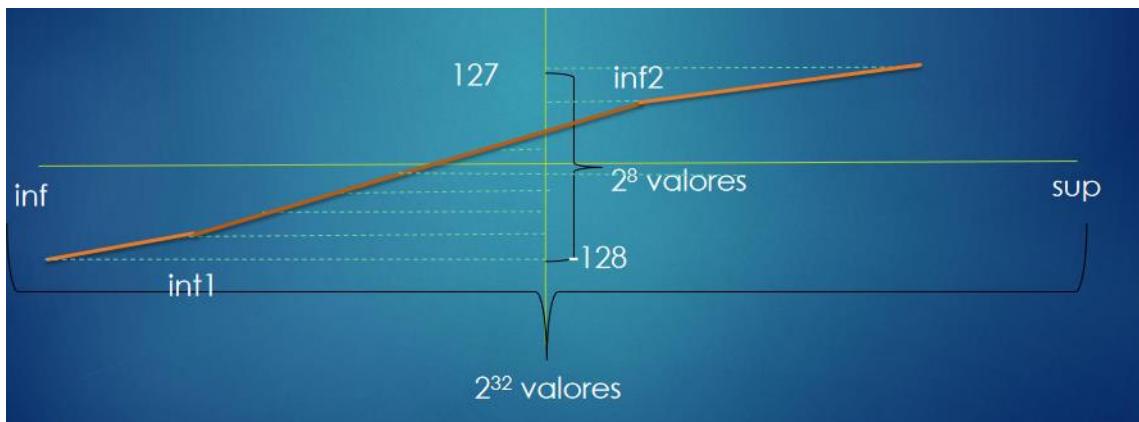
- También se pueden aplicar técnicas de cuantificación de rangos dinámicos e igualación de canales.

### Optimización de rangos dinámicos

La optimización de rangos dinámicos implica ajustar los rangos de cuantificación de los pesos y activaciones de la red en función de las estadísticas de los datos de entrada. Esto permite adaptar los rangos de cuantificación a los datos específicos utilizados durante la inferencia.

Una técnica común utilizada en la optimización de rangos dinámicos es la cuantificación por rangos lineales (linear range quantization). Esto implica dividir el rango de valores de los datos de entrada en varios subrangos y asignar diferentes rangos de cuantificación a cada subrango.

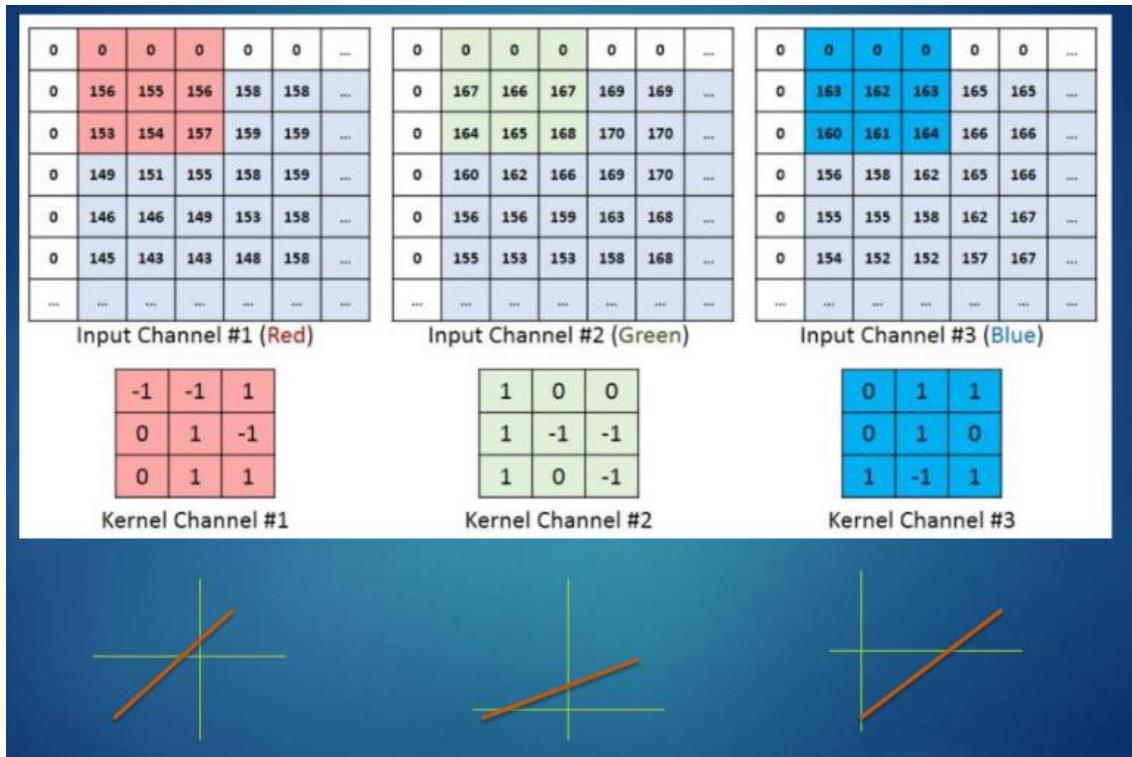
La optimización de rangos dinámicos es especialmente útil en aplicaciones donde los rangos de los datos de entrada pueden variar significativamente, como en el caso de imágenes o audio.



### Igualación de canales

La igualación de canales es una técnica que permite ajustar los rangos de cuantificación de los diferentes canales de una red neuronal por separado. Esto se hace asignando diferentes rangos de cuantificación a cada canal en función de sus características y requisitos específicos.

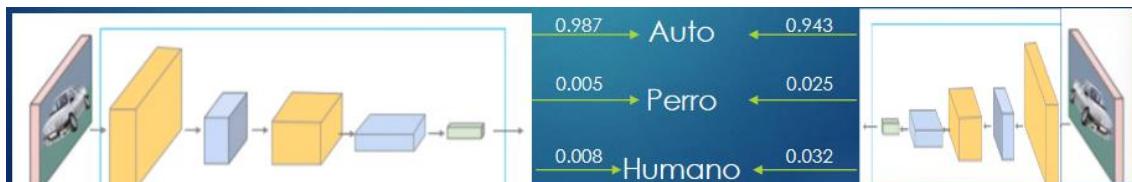
La igualación de canales es especialmente útil cuando los diferentes canales tienen diferentes rangos de valores y se desea optimizar los rangos de cuantización para cada canal de manera independiente.



### Cuantización posterior al entrenamiento

La cuantización se realiza en dos etapas

- Calibración: consiste en usar un conjunto de datos de muestra para determinar el rango de valores que se utilizará para cuantizar las activaciones.
- Cuantización: cada peso y activación se redondea al valor más cercano dentro del rango definido.
  - o Esto implica una pérdida de precisión.
  - o Se espera que la pérdida sea tolerable.



### Decuantificación

Cuando se usa el modelo se deben decuantificar los parámetros con el proceso inverso a la cuantificación.

La decuantificación puede ser:

- Total (ocupa lo mismo en memoria que el modelo original)
  - o No suele usarse
- Parcial
  - o El modelo ocupa menos memoria, pero hay que estar decuantificando los parámetros en cada inferencia.
    - Se aprovechan operaciones específicas del hardware

## Entrenamiento consciente

En el entrenamiento consciente (Quantization Aware Training) se lleva a cabo un entrenamiento de la red de alta precisión, emulando al mismo tiempo los parámetros de baja precisión.

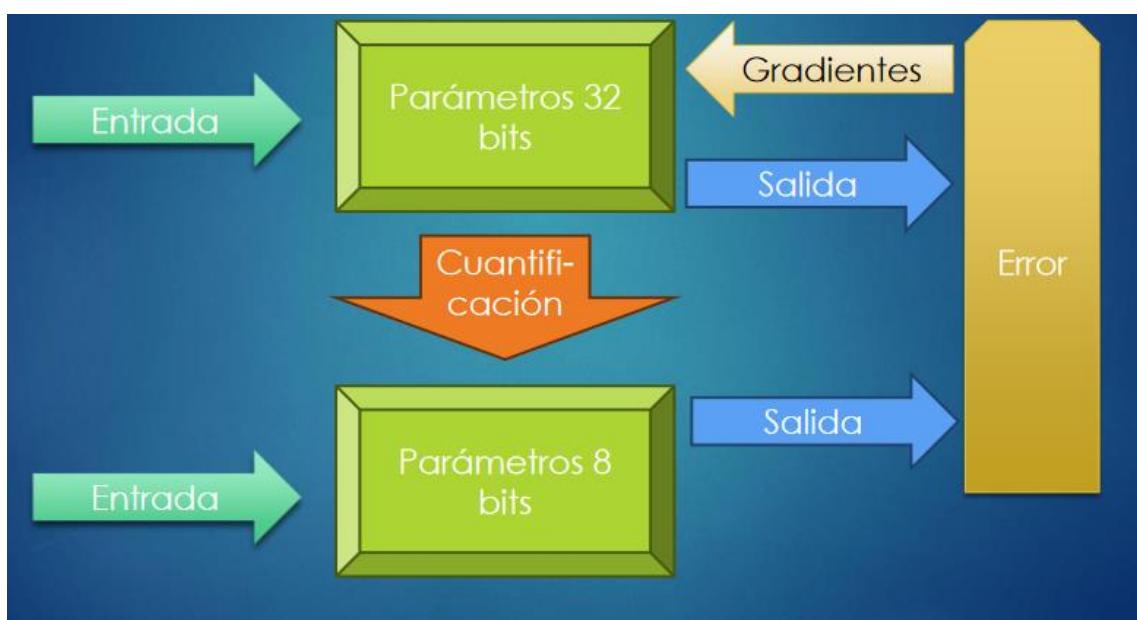
- Este entrenamiento suele ser un fine-tuning del modelo pre-entrenado.

Durante la fase forward, se usan los parámetros de alta precisión y los de baja precisión para calcular el error de la red.

Durante la fase backward los gradientes del error se calculan en alta precisión para actualizar los parámetros de alta precisión.

Luego se vuelven a cuantificar los parámetros.

La fase backward solo se realiza sobre el modelo de 32 bits



## TensorFlow Lite

TensorFlow Lite es un conjunto de herramientas que facilita la construcción de modelos en dispositivos móviles o de IoT.

Permite:

- Generar modelos “livianos”
- Convertir modelos existentes a su versión “liviana”

```
import tensorflow as tf
model = tf.keras.models.Sequential([ ... ])
model.compile(...)
model.fit(...)
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()
```

	<b>Mobilenet -v1-1-224</b>	<b>Mobilenet -v2-1-224</b>	<b>Inicio_v3</b>	<b>Resnet_v 2_101</b>
Precisión Top-1 (Original)	0.709	0.719	0.78	0.770
Precisión Top-1 (post-train)	0.657	0.637	0.772	0.768
Precisión Top-1 (aware-train)	0.7	0.709	0.775	N/A
Latencia (Original) (ms)	124	89	1130	3973
Latencia (post-train) (ms)	112	98	845	2868
Latencia (aware-train) (ms)	64	54	543	N/A
Tamaño (Original) (MB)	16.9	14	95.7	178.3
Tamaño (optimizado) (MB)	4.3	3.6	23.9	44.9

## Redes neuronales binarizadas

Los pesos y las activaciones ocupan un bit.

Solo tienen dos estados: -1 y 1

- BinaryNet (2016)
- BitNet (2018)
- BitNet (2023)
- 1-bit LLMs (2024)