Curso de Inteligencia Artificial para Ingenieros

Introducción a las Redes Neuronales Artificiales

Prof. Carlos A. Toro N.
carlos.toro.ing@gmail.com
2022

Objetivos

Al finalizar esta presentación, usted podrá:

- ❑ Describir la definición y desarrollo de las redes neuronales.
- ❑ Aprender acerca de los componentes importantes de las redes neuronales.
- ❑ Entender el proceso de entrenamiento y optimización de redes neuronales.
- ❑ Describir problemas comunes asociados a estas tecnologías.

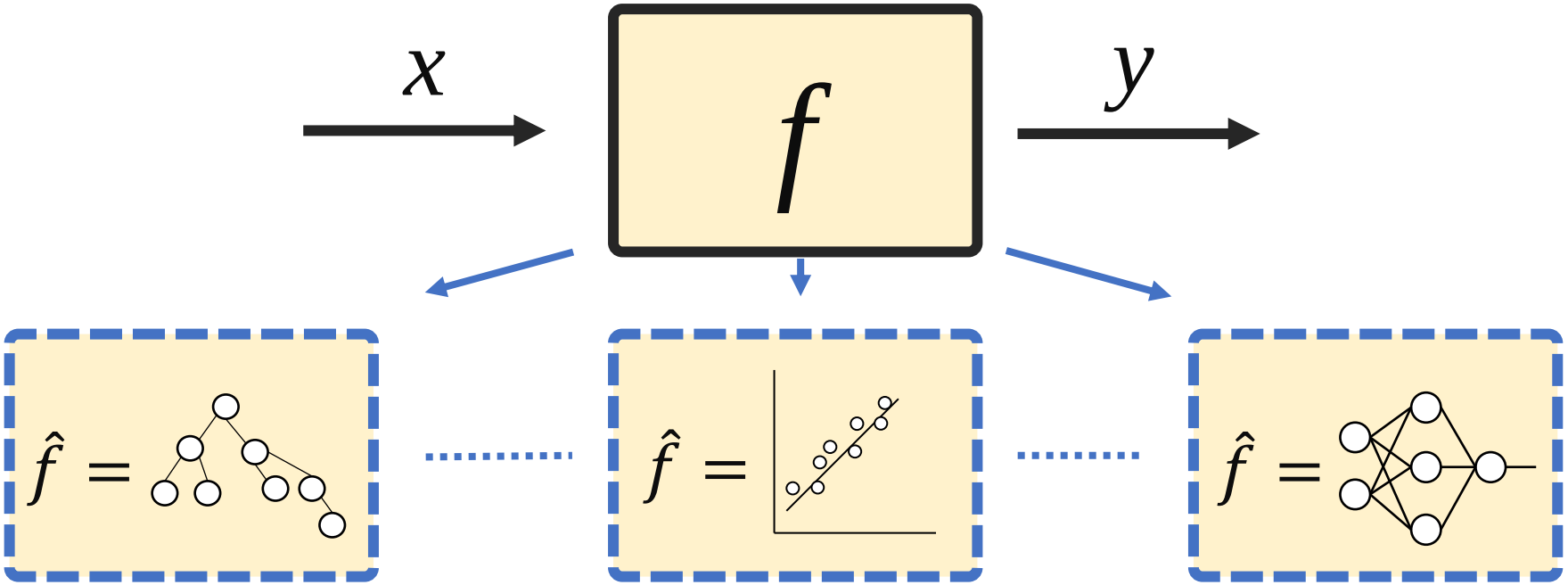
Contenidos

- Motivación: aprendizaje profundo
- Introducción a las redes neuronales artificiales
- Algoritmos para entrenar redes neuronales
- Problemas comunes y como solucionarlos
- Herramientas de programación: Tensorflow y Keras

Motivación: Aprendizaje profundo (*Deep Learning*)

Aprendizaje automático tradicional vs aprendizaje profundo

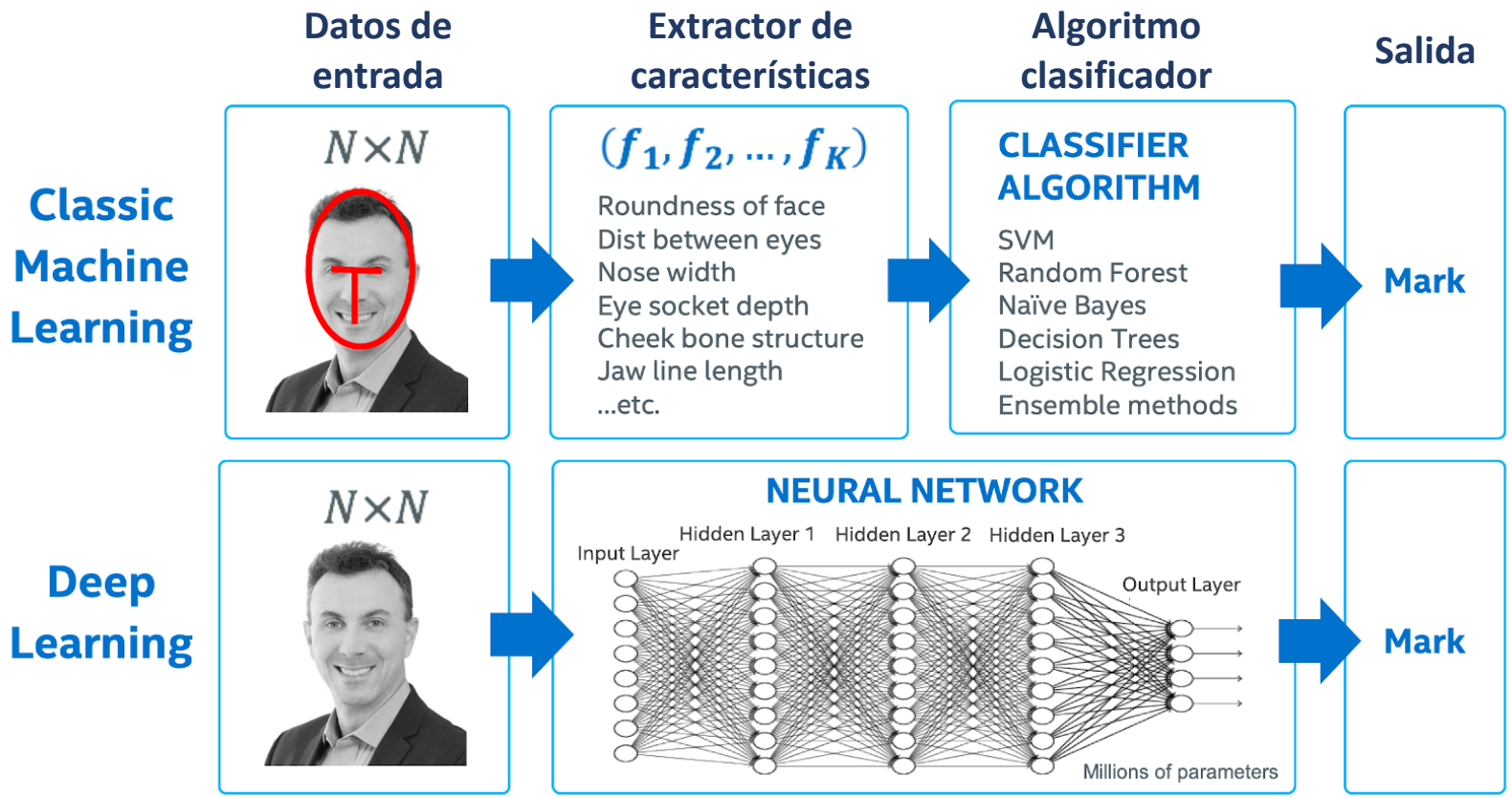
- Recordemos que al final del día, estamos tratado de encontrar una **función (modelo, algoritmo)** que asocie a cada elemento de x de un conjunto X a un elemento y de un conjunto Y:



Tenemos diferentes formas de encontrar esta función f o una aproximación de ella, ya hemos visto algunas relacionadas al machine learning en el curso, ahora corresponde ver otras basadas en algoritmos como redes neuronales artificiales.

Aprendizaje automático tradicional vs aprendizaje profundo

- El Aprendizaje Profundo o Deep learning son modelos basados en redes neuronales artificiales que aprenden representaciones jerárquicas composicionales de los datos de entrada y extraen de manera automática las características desde los datos:



Fuente imagen

Generalmente, la arquitectura de aprendizaje profundo está compuesto por una red neuronal profunda. “Profundo” en “Aprendizaje Profundo” se refiere al número de capas de una red neuronal.

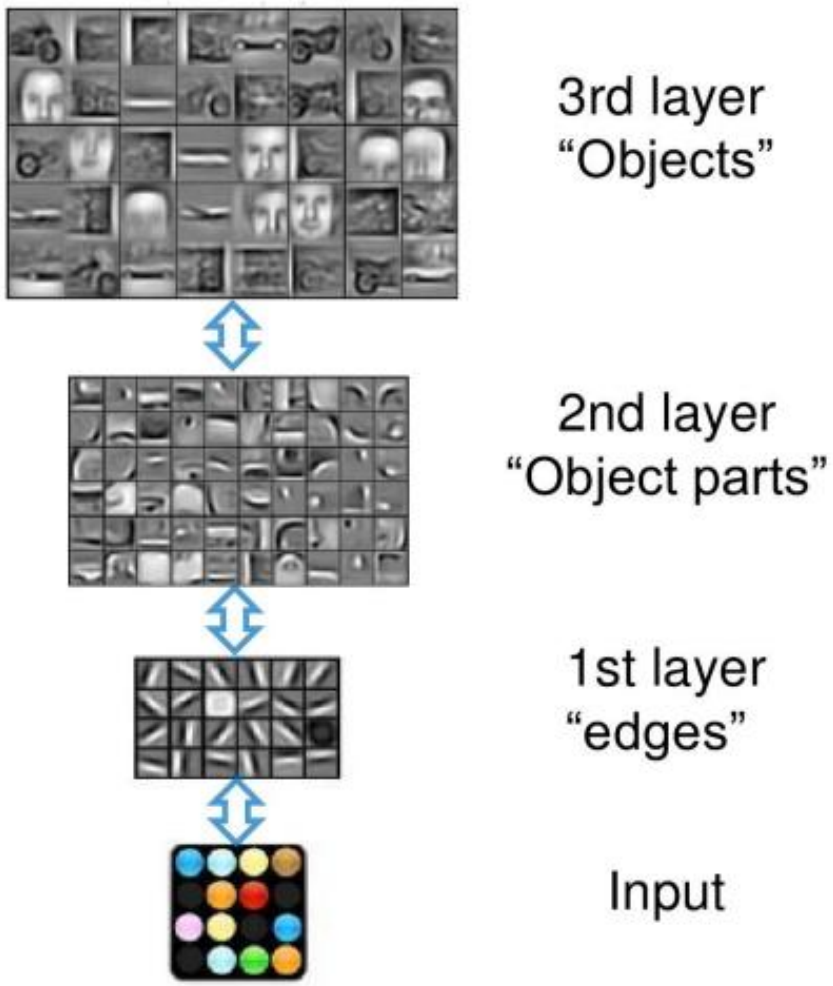
Aprendizaje automático tradicional vs aprendizaje profundo

- El Aprendizaje Profundo tiene grandes ventajas en campos como visión por computador, reconocimiento de voz y procesamiento de lenguaje natural, ya que son tareas donde extraer manualmente las características no es una tarea simple.

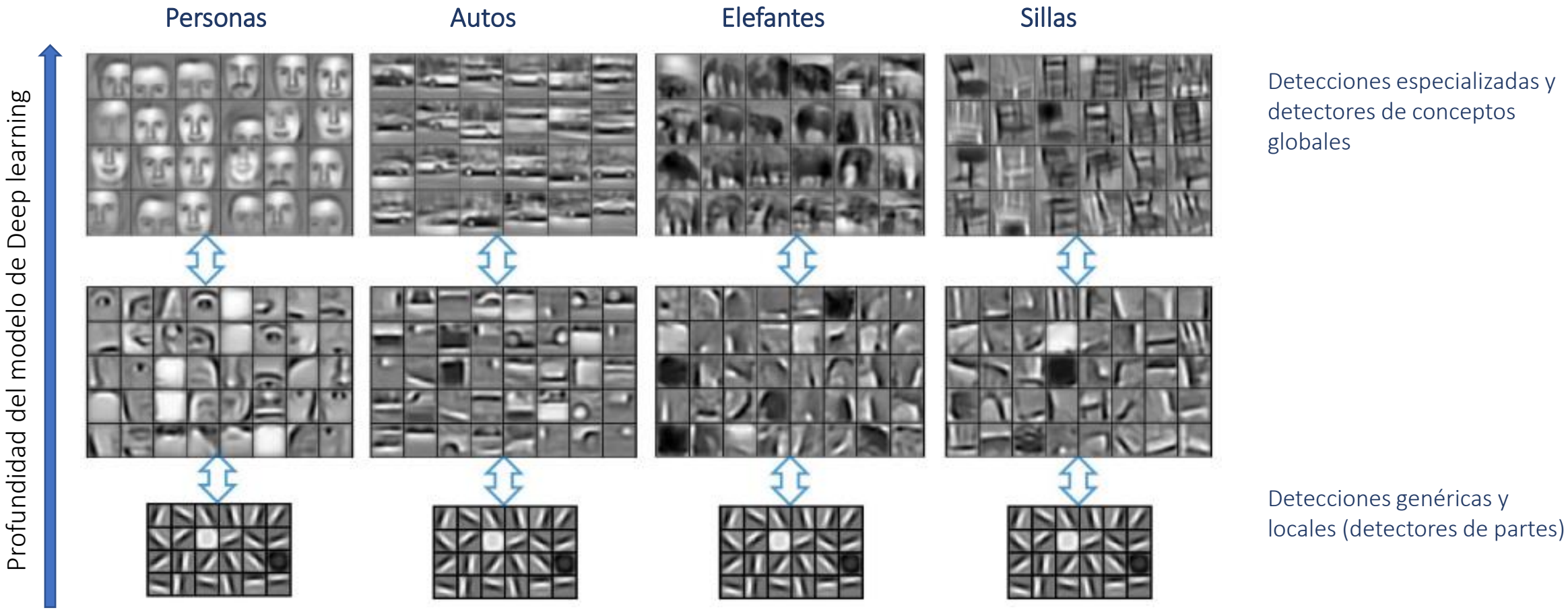
- Además de la extracción automática de características, la representación jerárquica y composicional es clave. El lenguaje es composicional, la visión es composicional, etc.

- Ejemplos:

  - Sonidos → fonemas → sílabas → palabras
  - Pixeles → bordes → partes → objetos
  - Caracteres → palabras → grupos de palabras → oraciones → textos



Aprendizaje automático tradicional vs aprendizaje profundo



Características extraídas en diferentes niveles de profundidad de un modelo de aprendizaje profundo.

¿Dónde debiéramos tener más características, a un nivel más bajo o más alto de jerarquía?

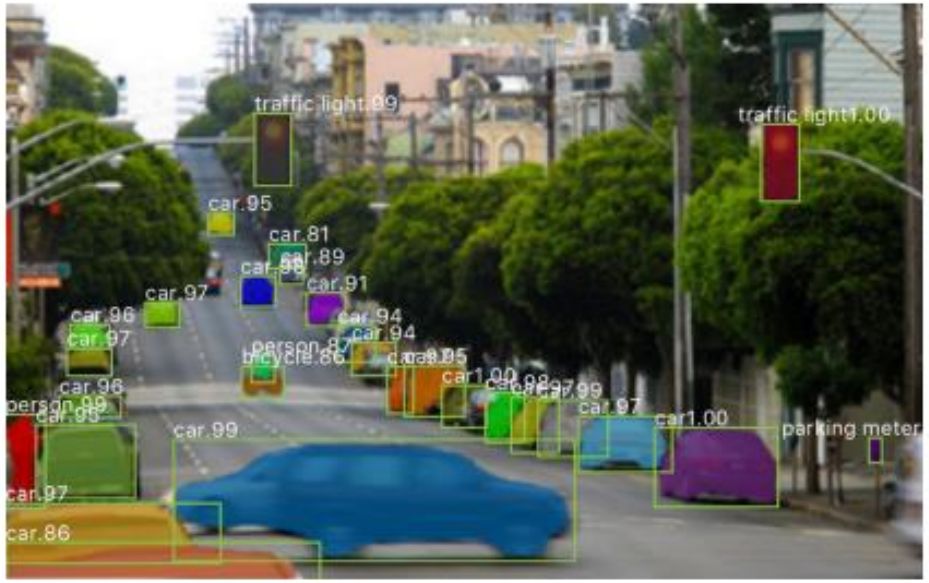
Aprendizaje automático tradicional vs aprendizaje profundo

- Algunas diferencias del *deep learning* con el *machine learning* tradicional son:

Machine Learning Tradicional	Deep Learning
Bajos requerimientos de hardware en el computador: no es necesario procesamiento paralelo, por ej. con GPUs. Esto puede beneficiar el procesamiento en sistemas electrónicos que operan con recursos de computo limitados.	Se requieren capacidades de cómputo mayores: Para ejecutar las operaciones matriciales en datos masivos, el computador necesita de GPUs para procesamiento paralelo.
Aplicable a entrenamientos con pequeñas cantidades de datos y cuyo desempeño pasado un punto no puede ser continuamente mejorado incluso si aumenta la cantidad de datos.	El desempeño puede ser altamente mejorado a medida que la complejidad de los modelos (n° de capas y pesos) aumenta y se tienen mayores cantidades de datos para el entrenamiento.
El problema se descompone en etapas	Aprendizaje extremo a extremo, E2E (End to End)
Extracción manual de características	Extracción automática de características dada por los algoritmos implementados.
Fácil de interpretar las características y decisiones tomadas por los modelos.	Interpretación más compleja de lo que aprendió el modelo para tomar una decisión.

Aprendizaje profundo en acción

Detección y segmentación de objetos



Traducción

Input sentence:	Translation (PBMT):	Translation (GNMT):	Translation (human):
李克強此行將啟動中加總理年度對話機制，與加拿大總理杜魯多舉行兩國總理首次年度對話。	Li Keqiang premier added this line to start the annual dialogue mechanism with the Canadian Prime Minister Trudeau two prime ministers held its first annual session.	Li Keqiang will start the annual dialogue mechanism with Prime Minister Trudeau of Canada and hold the first annual dialogue between the two premiers.	Li Keqiang will initiate the annual dialogue mechanism between premiers of China and Canada during this visit, and hold the first annual dialogue with Premier Trudeau of Canada.

Generación de video a partir de texto

Meta AI Make a Video model

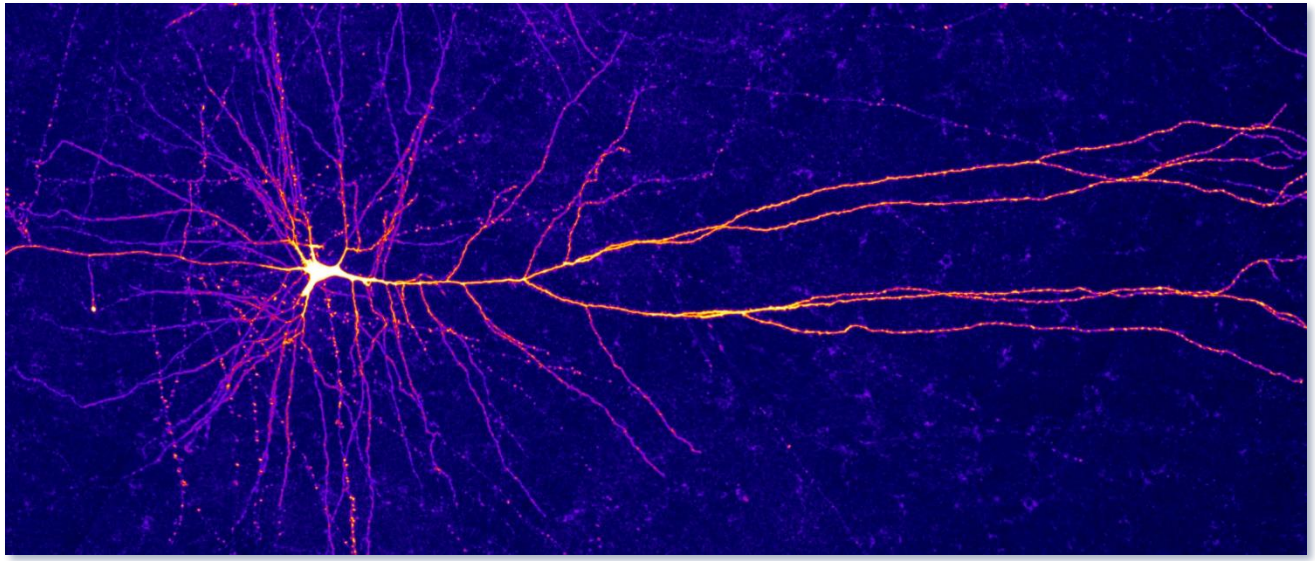
"A dog wearing a Superhero outfit with red cape flying through the sky"

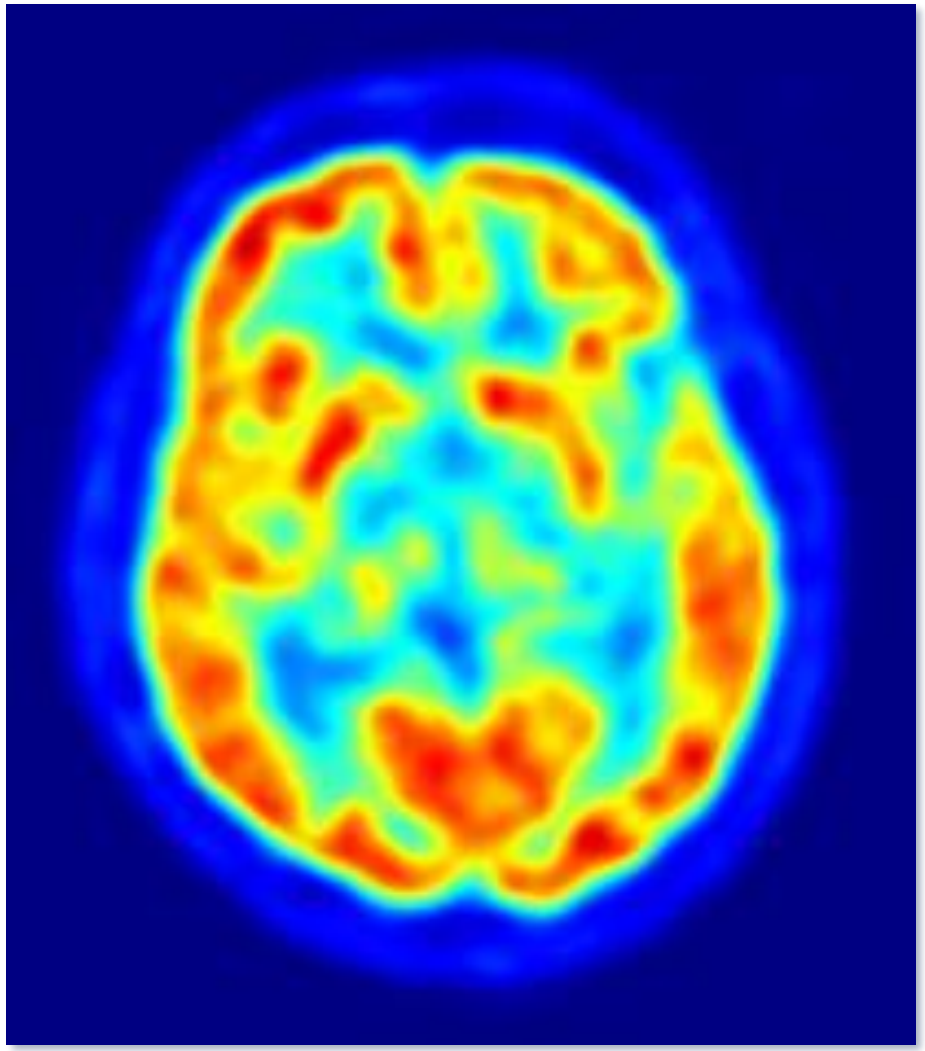
[https://makeavideo.studio/]

Introducción a la Redes Neuronales Artificiales

Introducción a las Redes Neuronales Artificiales (ANN)

- Las arquitecturas de ANN son motivadas por modelos de nuestro cerebro y células nerviosas.
- Las ANN son sistemas paralelos masivos con arreglos densos de procesadores simples.
- Mas interconexiones que número de procesadores/neuronas.
- Muchas aplicaciones de ANN pueden ser consideradas como tareas de mapeo de patrones, i.e., problemas de clasificación.





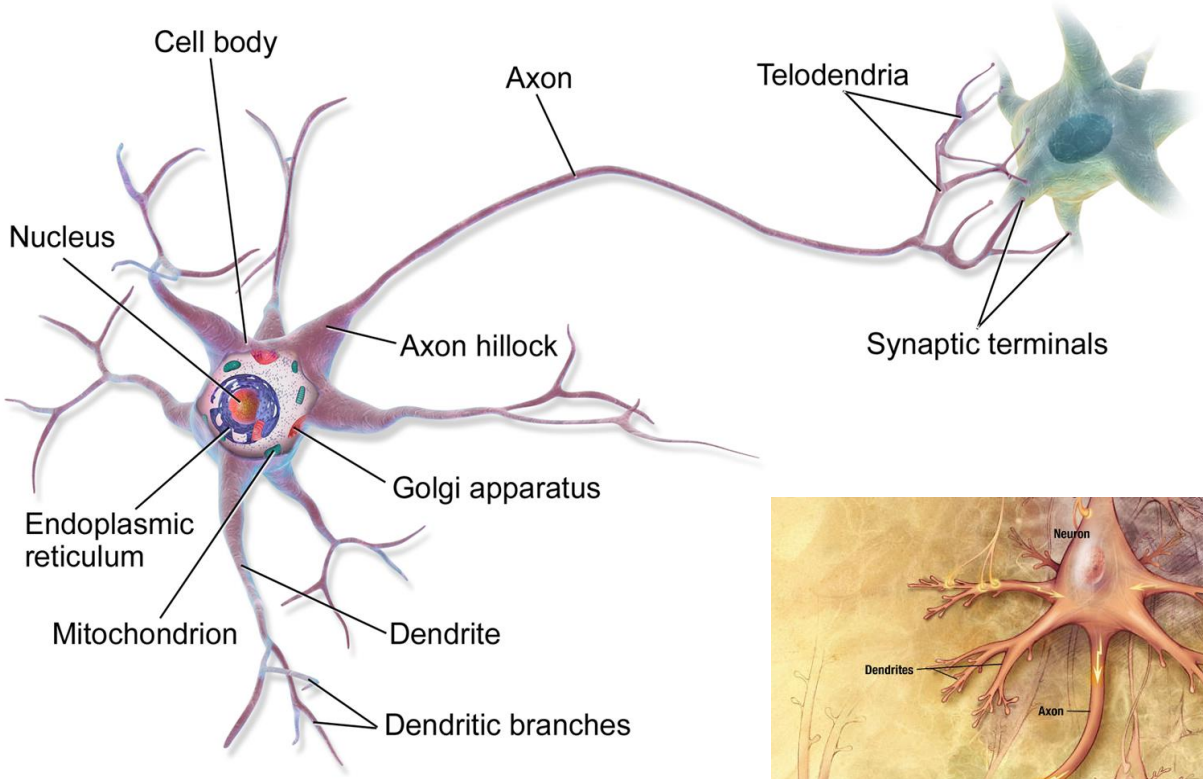
Introducción a las ANN

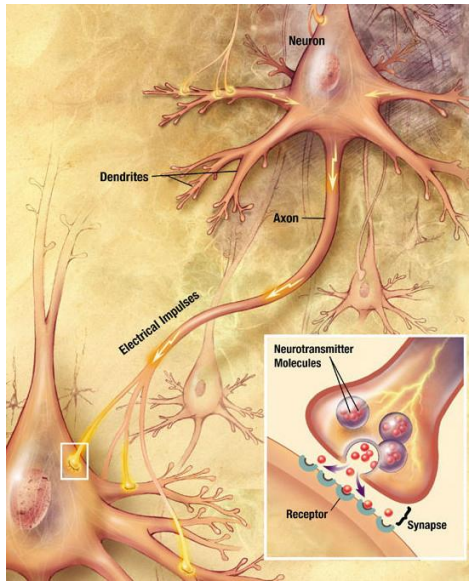
- En el cerebro humano, existen aproximadamente 10^{11} neuronas con 10^{15} interconexiones.

- Una neurona consiste de tres secciones:
  - Cuerpo de la célula (soma): Adición – disparo.
  - Dendritas: Reciben la señal desde otras células
  - Axón: Pasan la señal (pulsos) a otras células. Pueden medir desde ~ 0.1mm hasta 1 metro de longitud.

- El punto de conexión es conocido como sinapsis.

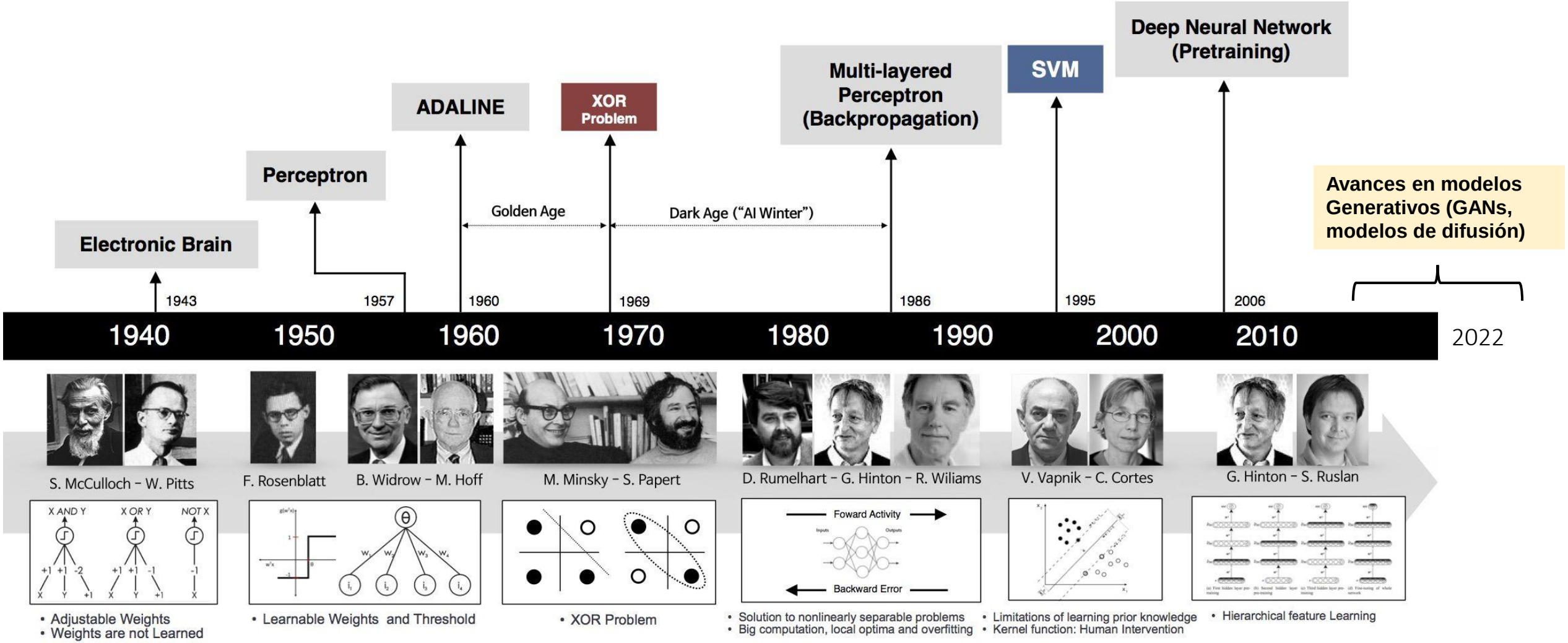
- Las señales recibidas por el cuerpo de la célula son sumadas y cuando esta suma (potencial) excede un umbral la célula genera un disparo, enviando un pulso a través del axón hacia otras neuronas.





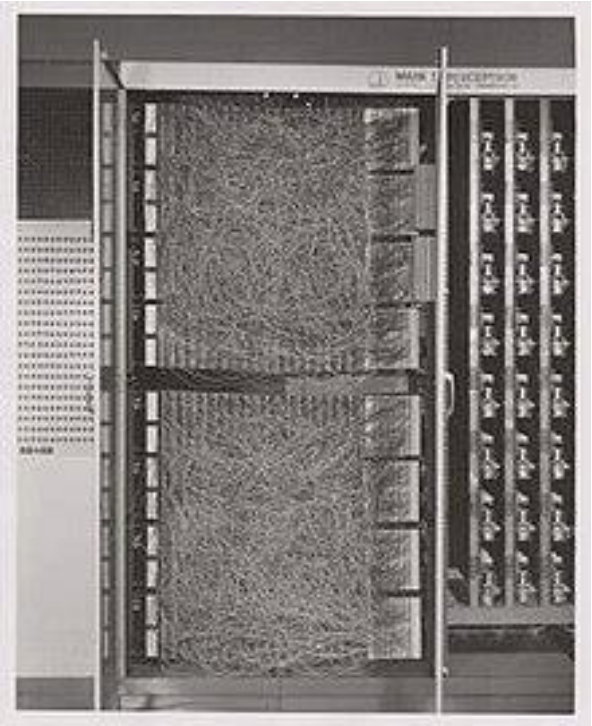
Fuente: wiki

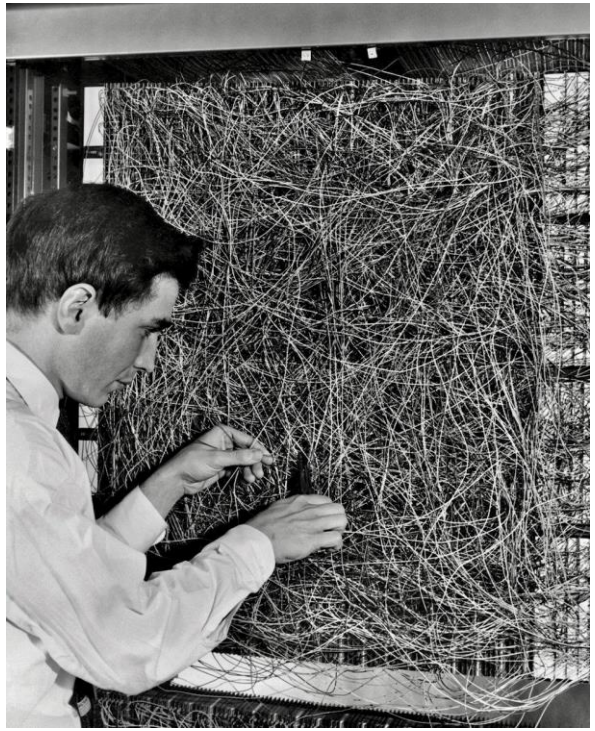
Desarrollo histórico de las redes neuronales artificiales



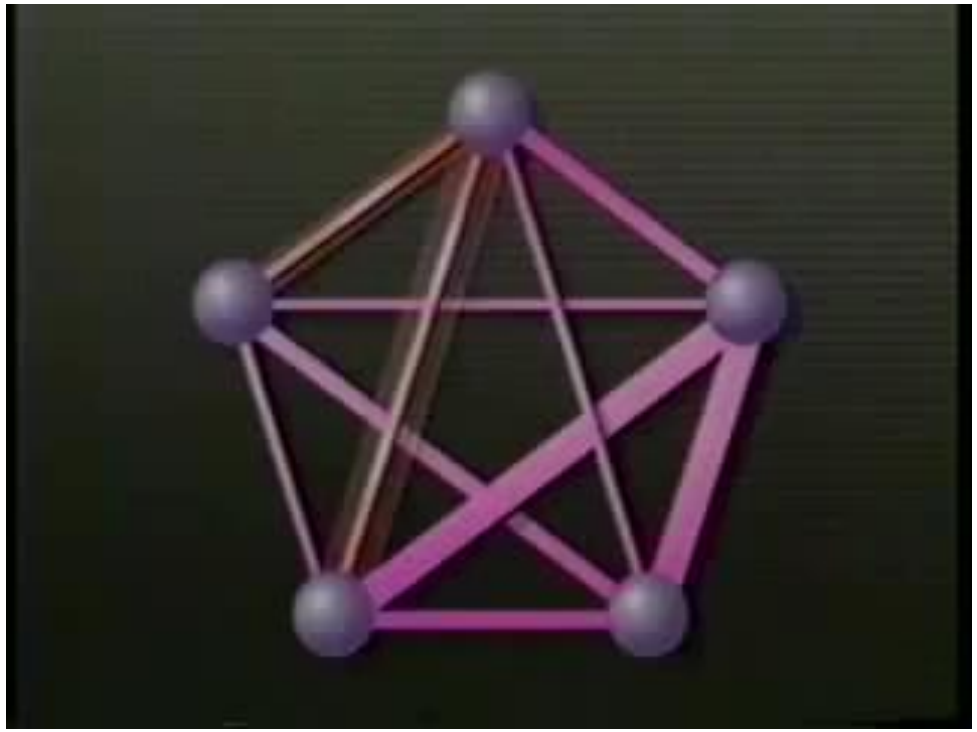
Desarrollo histórico de las redes neuronales artificiales

- En esta curso nos enfocaremos en técnicas de aprendizaje supervisado con redes neuronales.
- En particular, en esta clase, en estructuras del tipo superficial (*shallow*) como el perceptrón multicapa.
- Pero, qué es el perceptrón?: un clasificador en su origen y la base de todos los algoritmos de redes neuronales actuales.



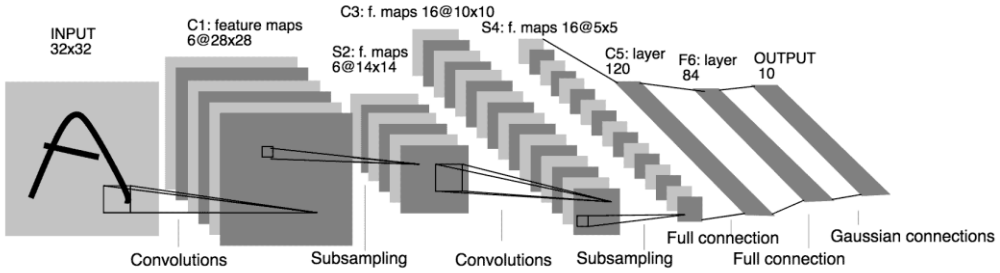


Frank Rosenblatt –1950-60

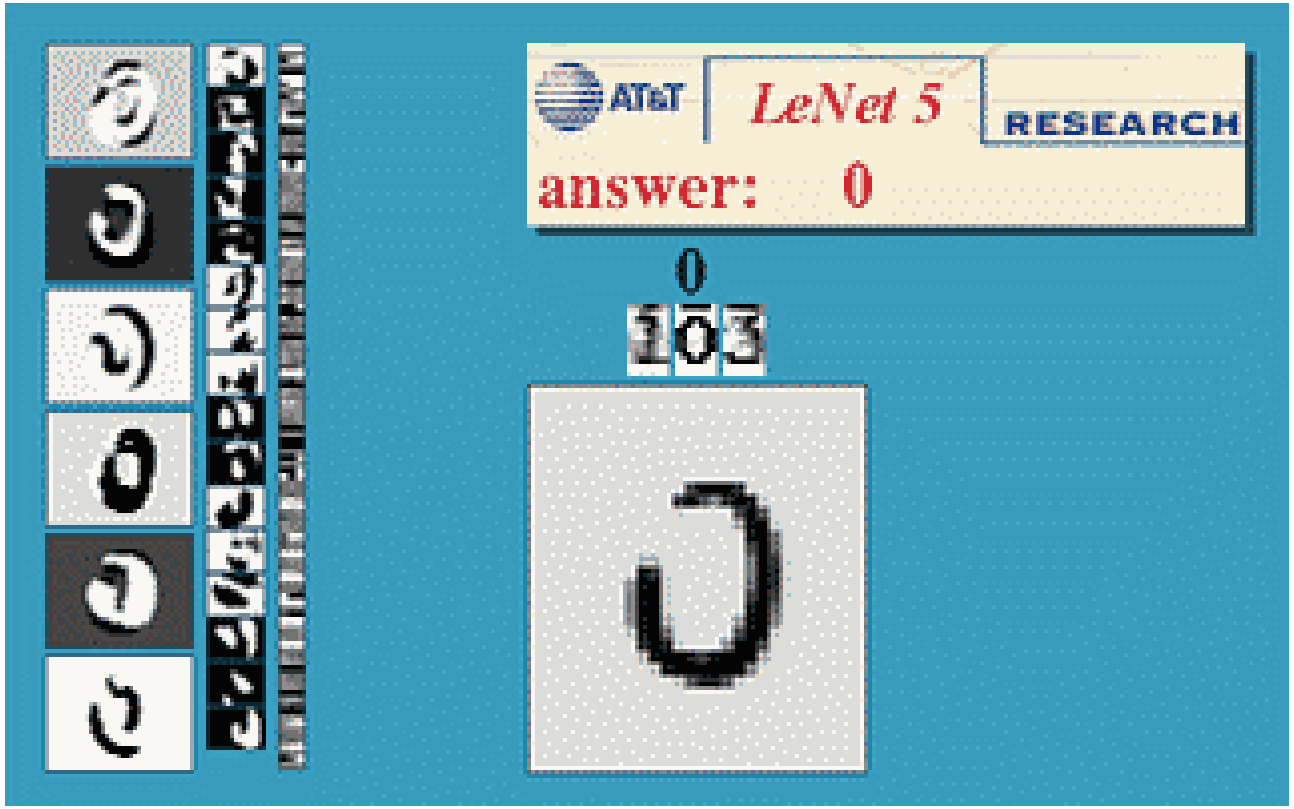


https://www.youtube.com/watch?v=cNxadbrN_aI

Desarrollo histórico de las redes neuronales artificiales



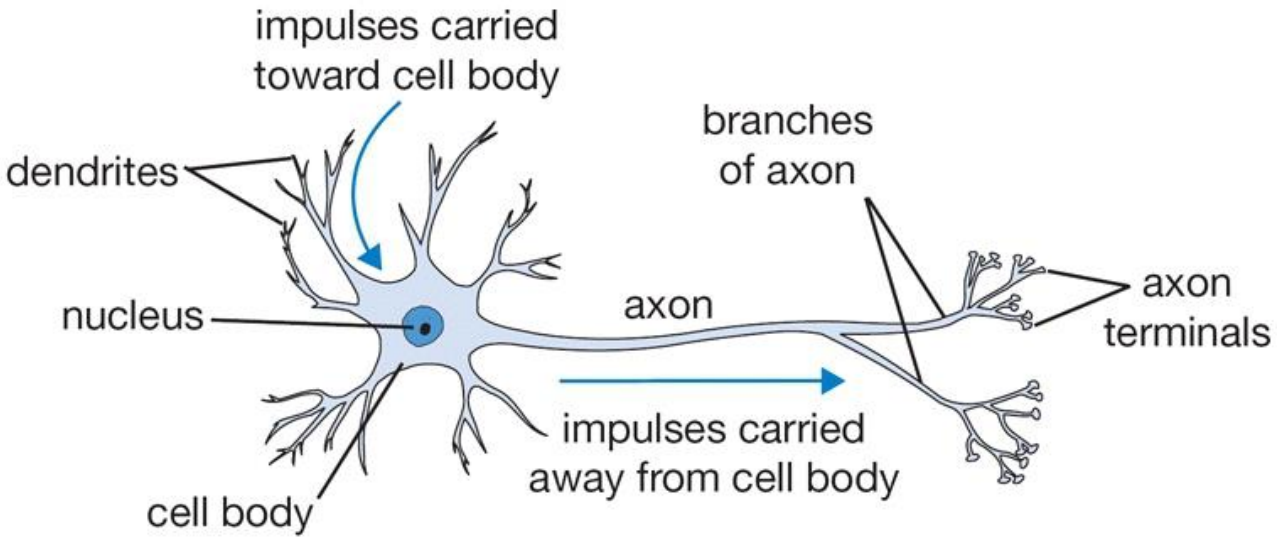
Yann LeCun, LeNet5 (1998)



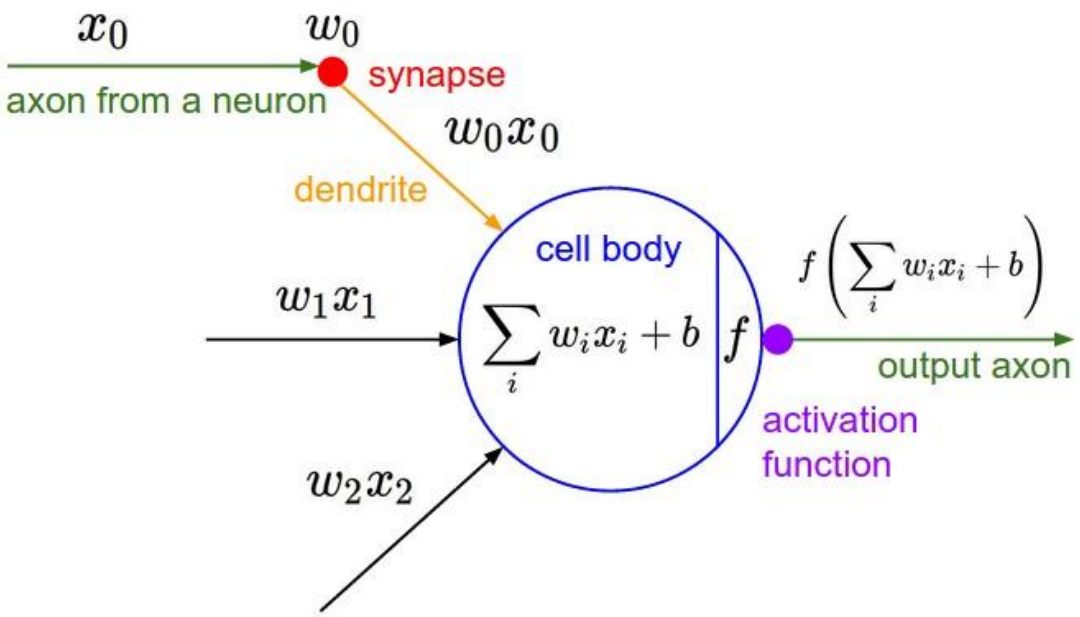
http://yann.lecun.com/index.html

Introducción a las Redes Neuronales Artificiales

Neurona biológica



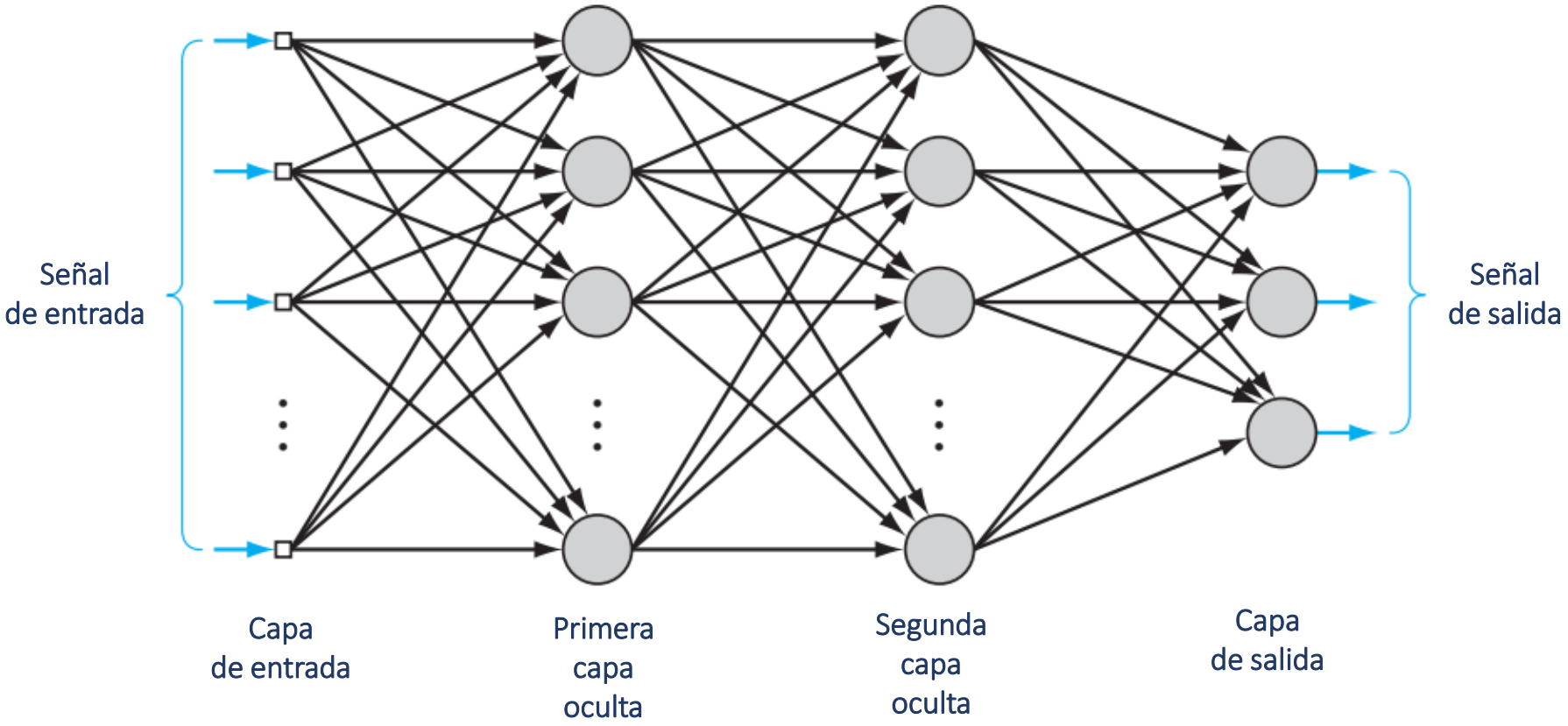
Perceptrón o neurona artificial



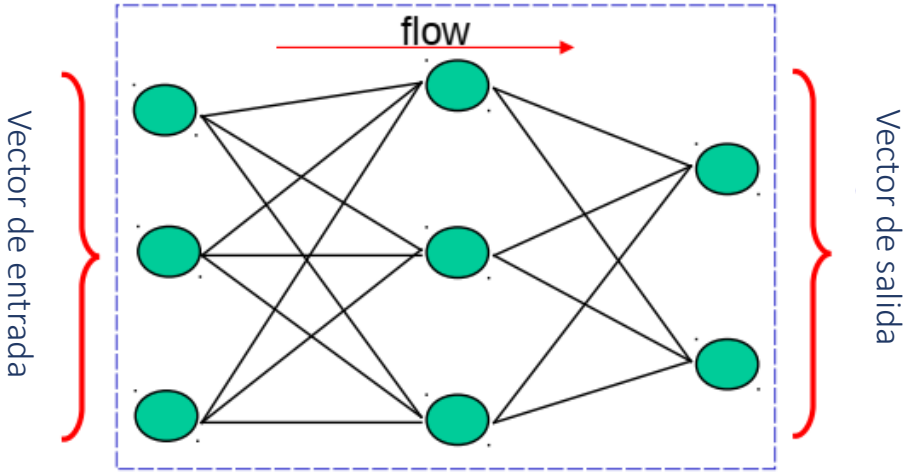
Fuente

Introducción a las Redes Neuronales Artificiales

- Ej. Perceptrón Multicapa con dos capas ocultas:

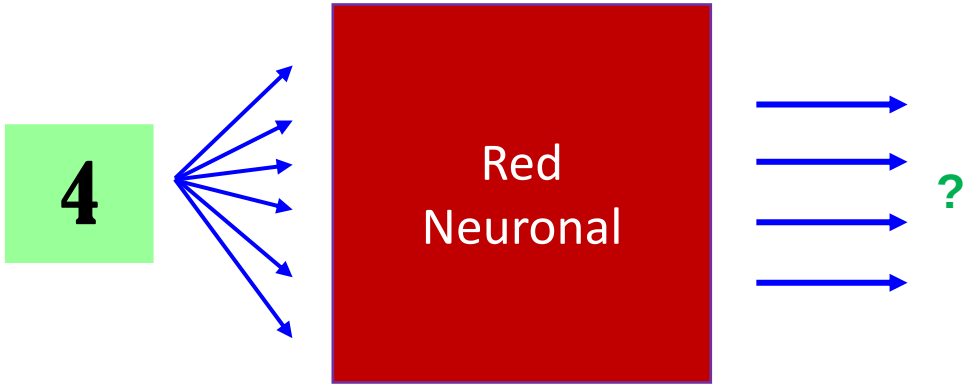


Introducción a las Redes Neuronales Artificiales



OBJETIVO: Dada la **estructura** de la ANN, necesitamos **ajustar los pesos** en la red, tal que ellos modelen las relaciones entrada-salida en el set de **datos de entrenamiento**.

Ejemplo:
Cuando la ANN recibe una imagen del dígito 4, debiera ser capaz de generar un código de salida que indique que es un cuatro.



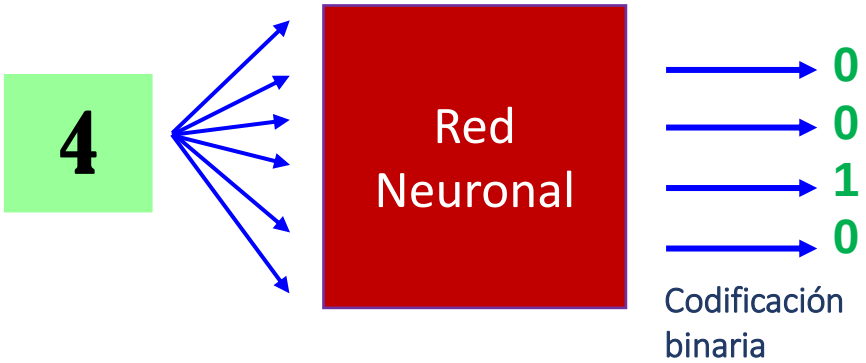
Introducción a las Redes Neuronales Artificiales

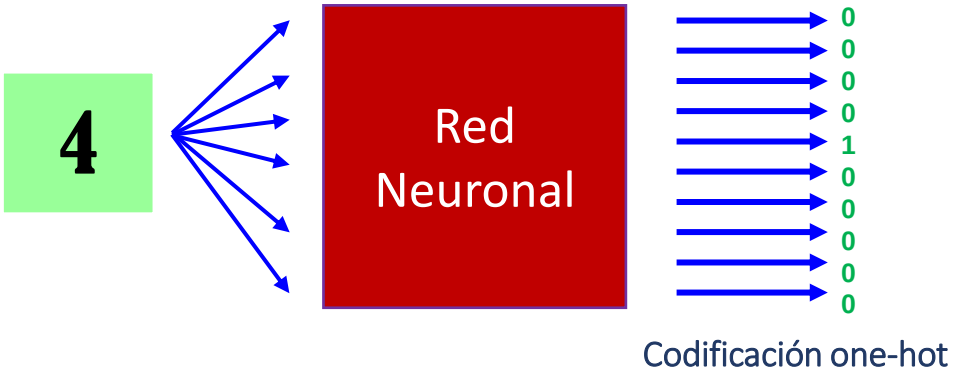
¿Cómo ajustamos el número de neuronas en cada capa?

Capa de entrada: en general, el número de unidades en la entrada es igual al número de variables o características de entrada.

Capa oculta: el número de unidades en la(s) capa(s) oculta(s) (y el número de capas ocultas), depende de la complejidad del problema. Problemas de clasificación o predicción más difíciles requieren de estructuras más complejas para alcanzar el objetivo.

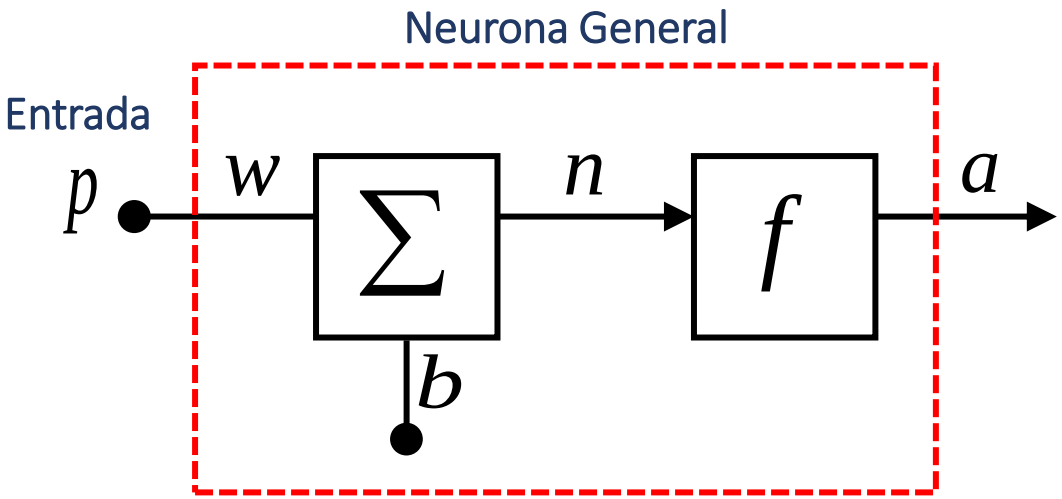
Capa de salida: el número de unidades en la capa de salida depende de la codificación seleccionada. Por ejemplo, consideremos los siguientes casos en un problema de clasificación:





El perceptrón

- Ejemplo de una neurona (perceptrón) con un solo elemento de entrada



Entrada : $p \in \mathbb{R}$

Parámetros :

$Peso : w \in \mathbb{R}$

$bias : b \in \mathbb{R}$

Campo local : $n \in \mathbb{R}$

Función de activación : $f : \mathbb{R} \rightarrow \mathbb{R}$

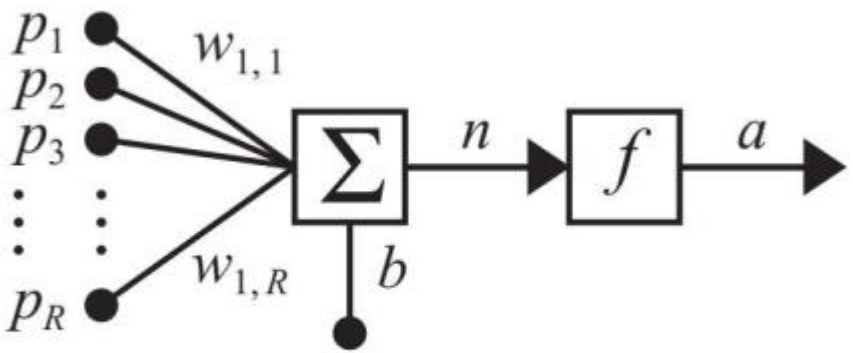
Salida : $a \in \mathbb{R}$

$$a = f(w \cdot p + b)$$

El perceptrón

- Para un vector de entradas (ejemplos con varias variables de entrada)

Representación por elementos



$$a = f(n)$$

Campo local : $n = w_{1,1}p_1 + w_{1,2}p_2 + \cdots + w_{1,R}p_R + b$

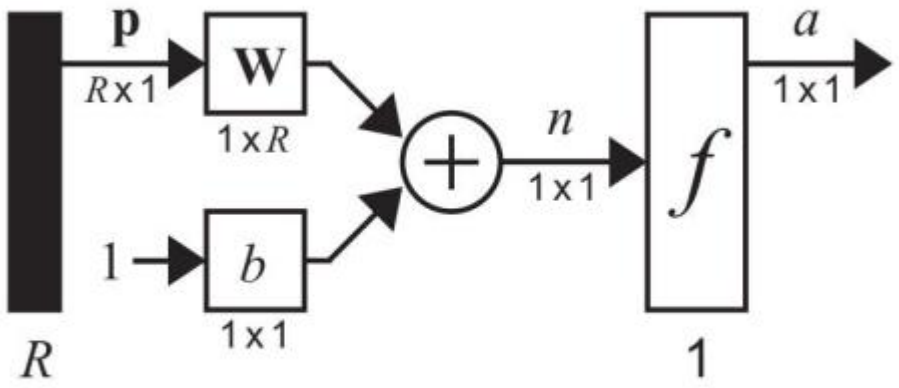
Pesos : $w_{1,1},\ w_{1,2},\ \ldots,\ w_{1,R}$

Bias : b

Entradas : $p_1,\ p_2, \ldots,\ p_R$

Número de Entradas : R

Representación vectorial



$$a = f(\mathbf{W}\boldsymbol{p} + b)$$

Campo local : $n = \boldsymbol{W}\boldsymbol{p} + b$

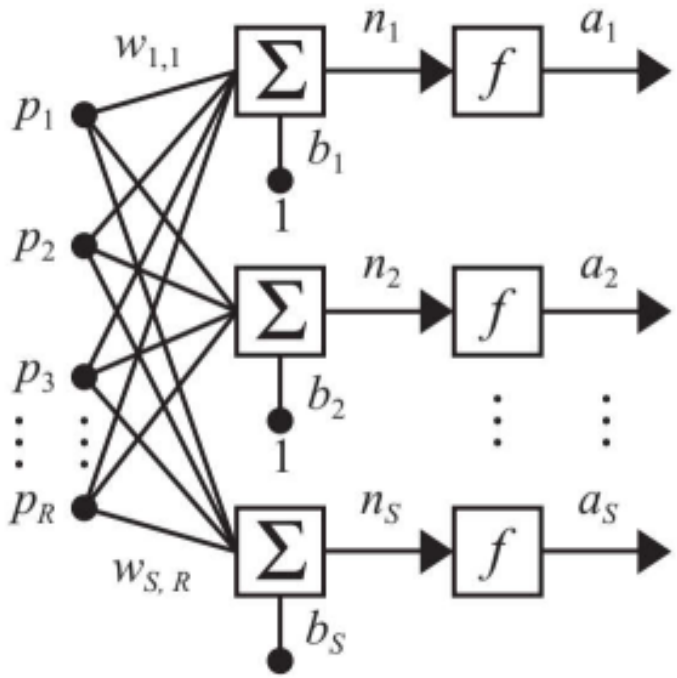
Vector de Pesos : $\boldsymbol{W} = [w_{1,1}\ \ w_{1,2}\ \ \ldots\ \ w_{1,R}]$

Bias : b

Entradas : $\boldsymbol{p} = [p_1\ \ p_2\ \ \ldots\ \ p_R]^T$

El perceptrón

- Red Neuronal de una capa con varias neuronas:



$$a_1 = f(w_{1,1}p_1 + w_{1,2}p_2 + \cdots w_{1,R}p_R + b_1)$$

$$a_2 = f(w_{2,1}p_1 + w_{2,2}p_2 + \cdots w_{2,R}p_R + b_2)$$

$$\vdots$$

$$a_S = f(w_{S,1}p_1 + w_{S,2}p_2 + \cdots w_{S,R}p_R + b_S)$$

Número de entradas: *R*

Número de salidas: *S*

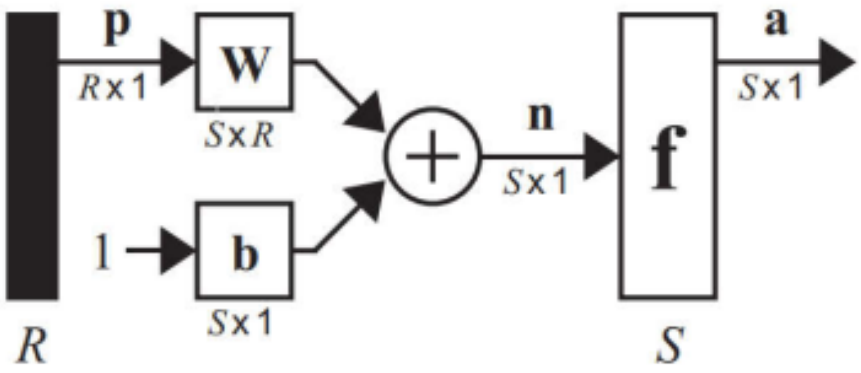
Notación:

$w_{i,j}$ = peso conectando la neurona *i* con la entrada *j*.

Recordar: Cada una de las entradas *R* está conectada a una neurona a través de un peso w_{ij}

El perceptrón

- Red Neuronal de una capa: forma vectorial



$$\boldsymbol{a} = \boldsymbol{f}(\mathbf{W}\boldsymbol{p} + \boldsymbol{b})$$

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix}$$

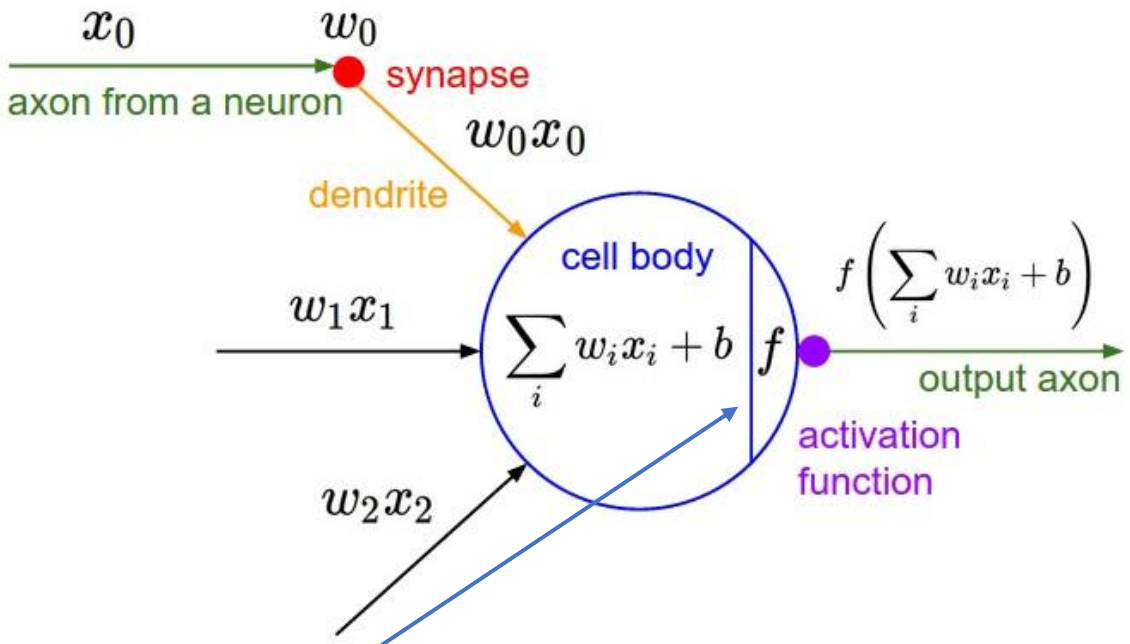
R = Número de entradas
S = Número de salidas

$$\mathbf{p} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_R \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_S \end{bmatrix} \quad \mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_S \end{bmatrix}$$

Funciones de Activación

Funciones de activación

- Las funciones de activación son importantes para los modelos de redes neuronales para aprender y entender funciones no lineales complejas. Estas permiten introducir características no lineales a los modelos.
- Sin las funciones de activación, las señales de salida son solo funciones lineales simples. La complejidad de las funciones lineales es limitada, y la capacidad de aprendizaje de funciones complejas desde los datos es baja.



Función de activación

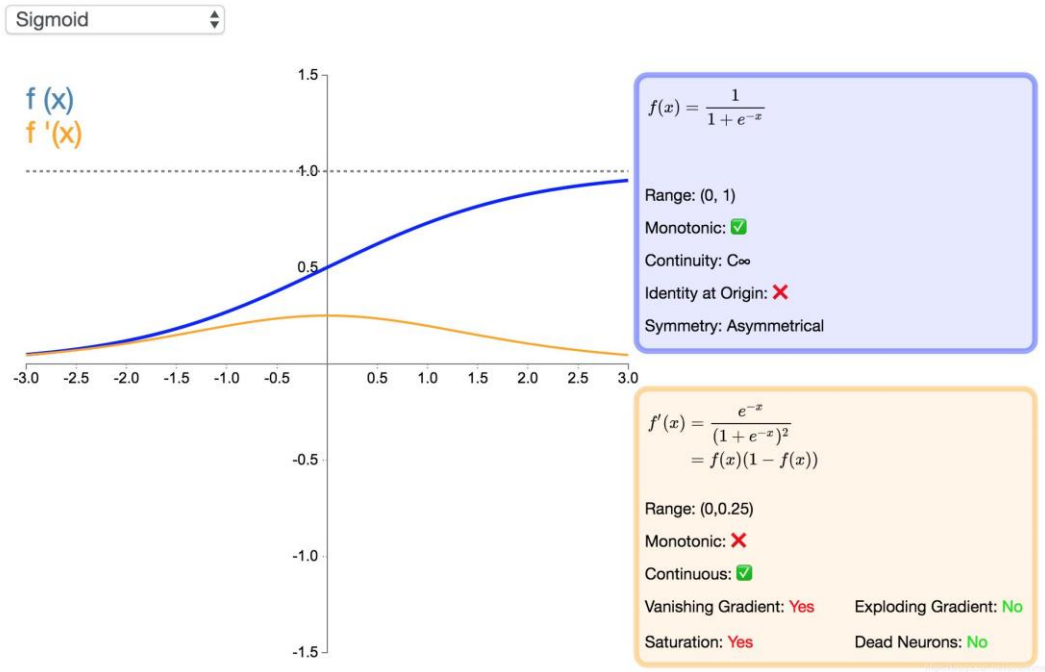
$$output = f(w_0 x_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 \ldots) = f(W^t \bullet X)$$

Acá referencia extensa de funciones de activación

Funciones de activación

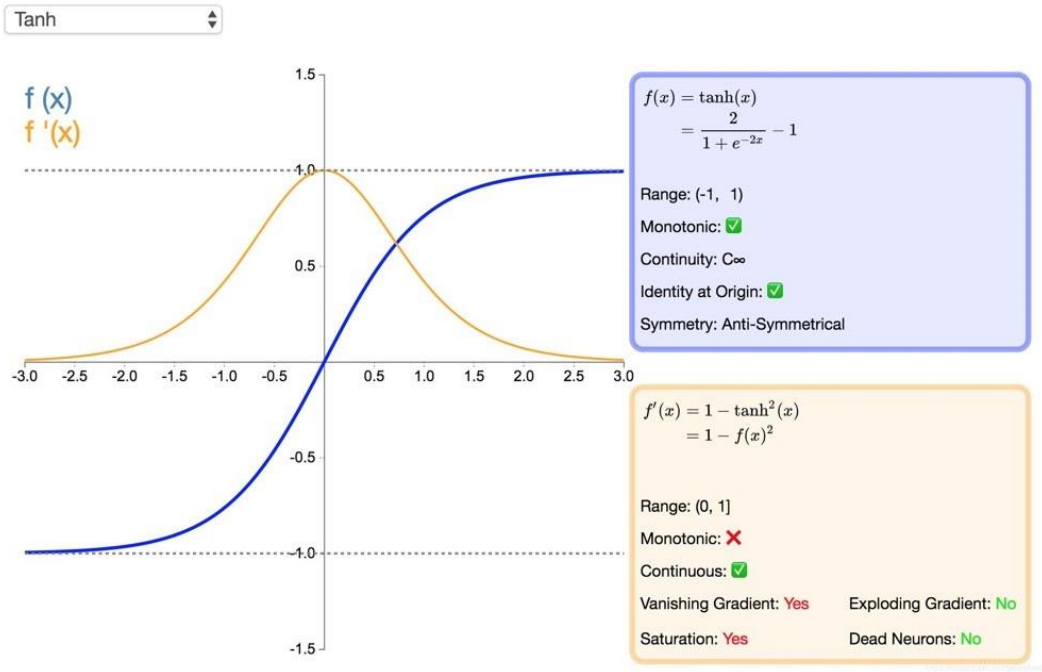
Sigmoidal

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



Tangente hiperbólica

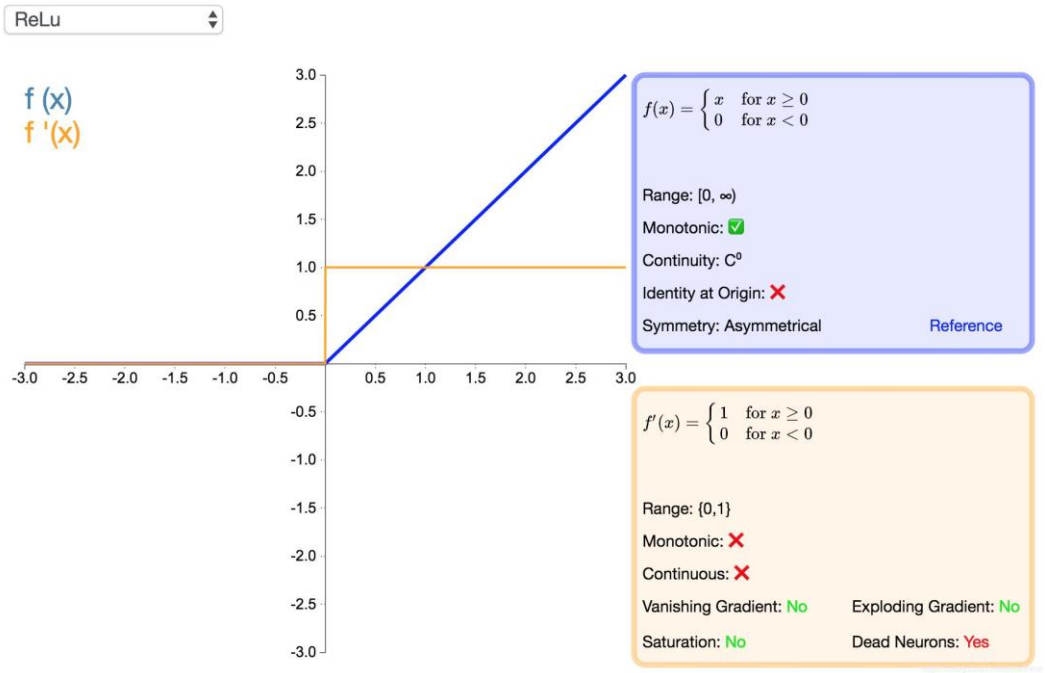
$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



Funciones de activación

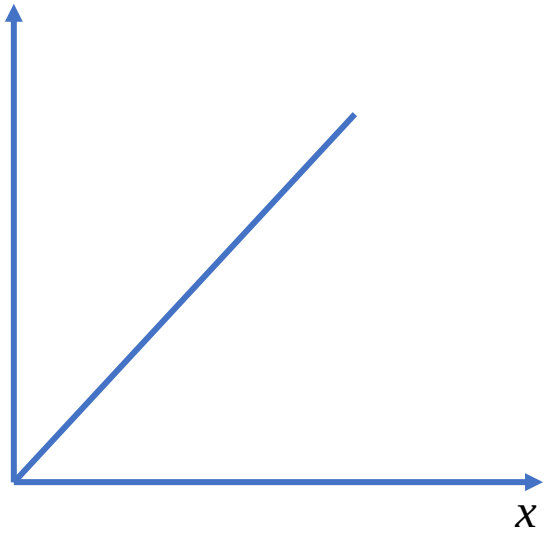
Unidad de Rectificación Lineal (ReLU)

$$y = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$$



Lineal (caso particular para la salida de redes en problemas de regresión)

$$y = f(x) = x$$



Funciones de activación

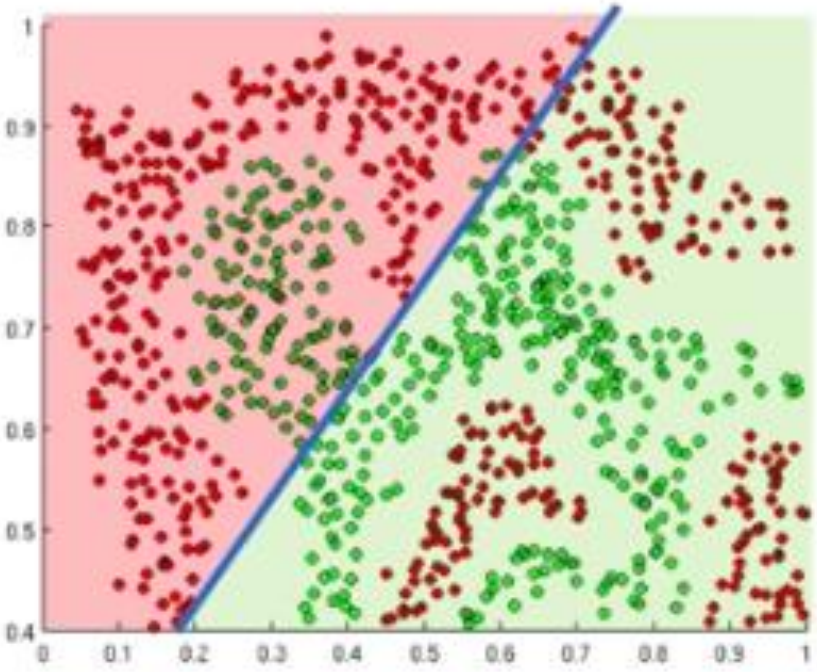
Función Softmax

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

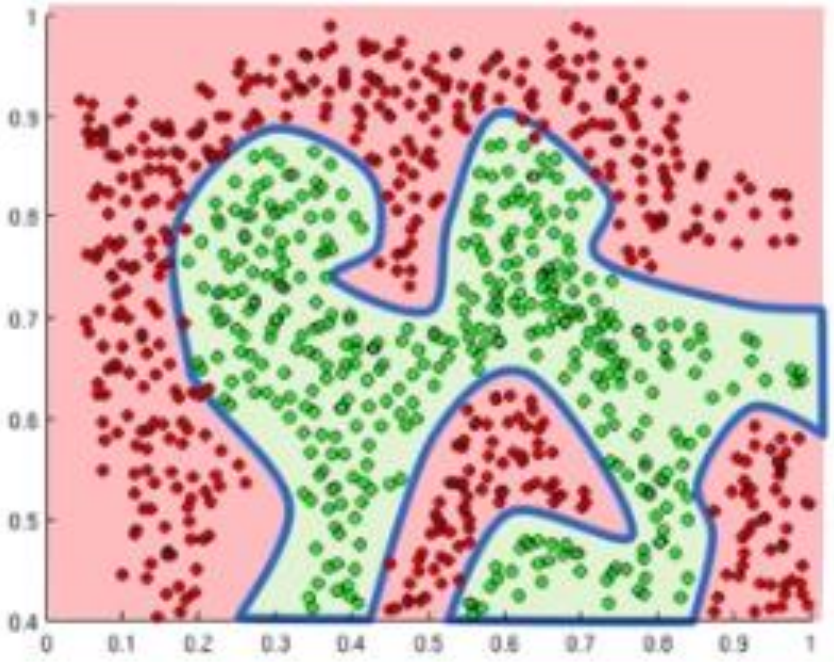
- La función Softmax es usada para mapear un vector K-dimensional de un número arbitrario de valores reales a un vector K-dimensional de valores reales, donde cada elemento del vector esta en el intervalo 0 a 1. Todos los elementos del vector resultante suman 1.
- La función Softmax es a menudo usada como capa de salida en una tarea de clasificación multiclase.

Funciones de activación

La importancia de las funciones de activación radica en que introducimos no-linealidades en la red que nos permiten aprender funciones arbitrarias, y no solo lineales.



Funciones de activación lineal producen decisiones lineales, sin importar el tamaño de la red.



Funciones de activación no lineales nos permiten aproximar funciones de complejidad arbitraria.

Cómo Aprenden las Redes Neuronales

() Recordemos: Qué son los gradientes

Sea $f := f(\boldsymbol{x})$, f: $\mathbb{R}^n \rightarrow \mathbb{R}$, $\boldsymbol{x} = [x_1, \ldots, x_n]$ una función/campo escalar, se define el operador gradiente de f en $\boldsymbol{x}$ como $\nabla f(\boldsymbol{x})$:

$$\nabla f(\boldsymbol{x}) = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right) = \left(f_{x_1}, \ldots, f_{x_n} \right)$$

- El gradiente se representa con el operador diferencial nabla ∇ seguido de la función.

- En este caso se genera un campo vectorial, donde el vector es perpendicular a las curvas de nivel

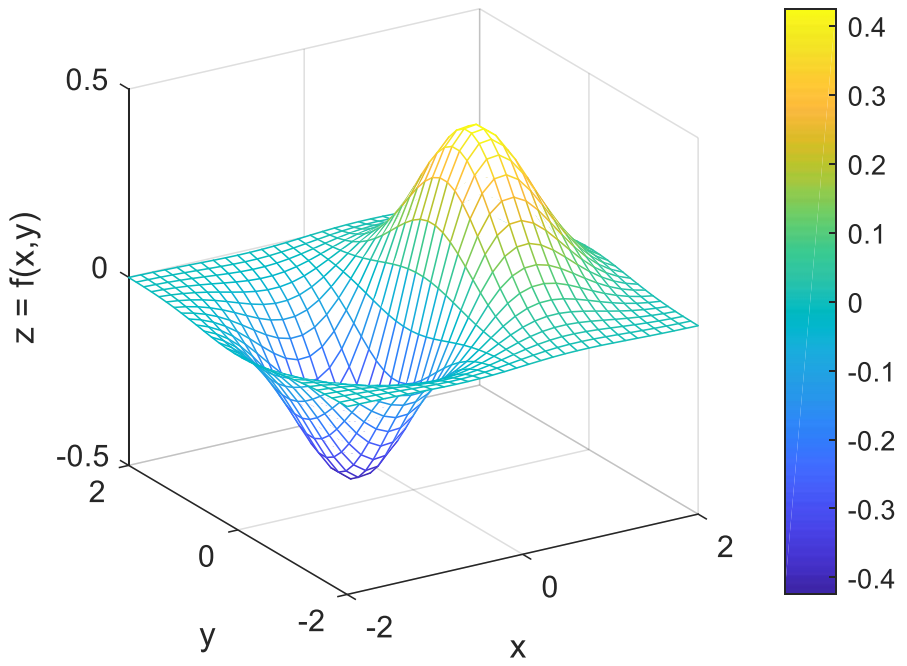
$$C_\lambda = \{\boldsymbol{x} \in \mathbb{R}^n \mid f(\boldsymbol{x}) = \lambda\}.$$

- Además, el gradiente indica la dirección de máximo crecimiento de f en el punto $\boldsymbol{x}$ y su módulo representa el ritmo de dicha variación en la dirección dada.

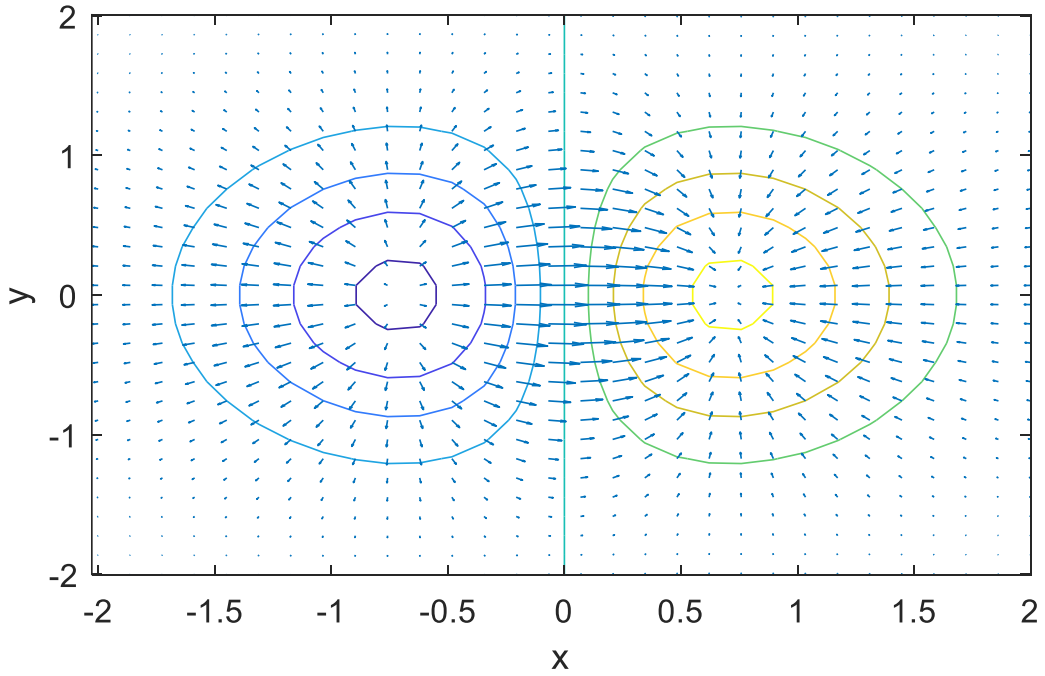
() Recordemos: Qué son los gradientes

Ejemplo: graficar la siguiente función escalar como una superficie y el correspondiente campo vectorial gradiente.

$$f(x,y) = xe^{-x^2-y^2}$$



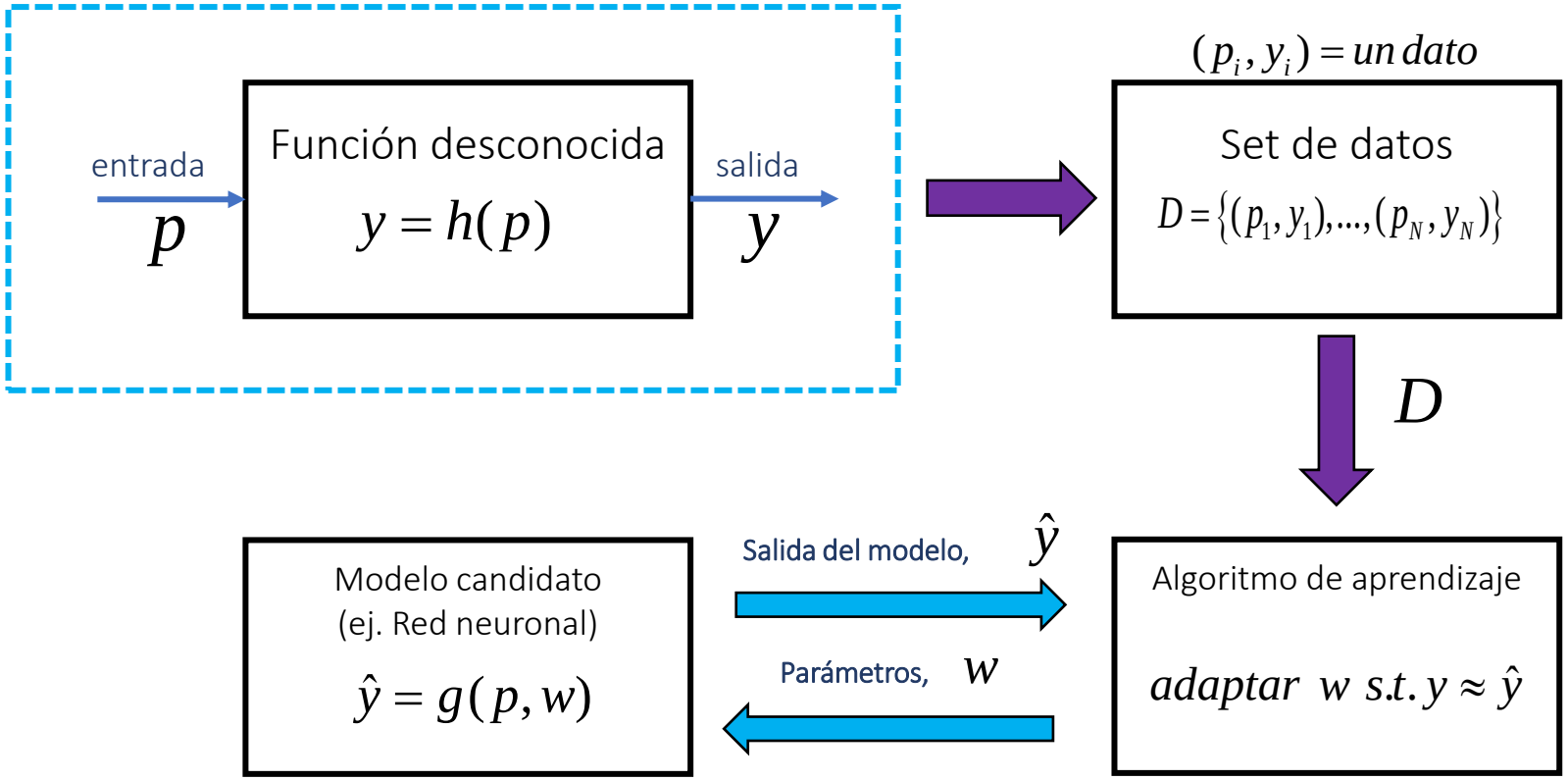
Función escalar.



Campo vectorial gradiente.

Proceso de entrenamiento de una red neuronal

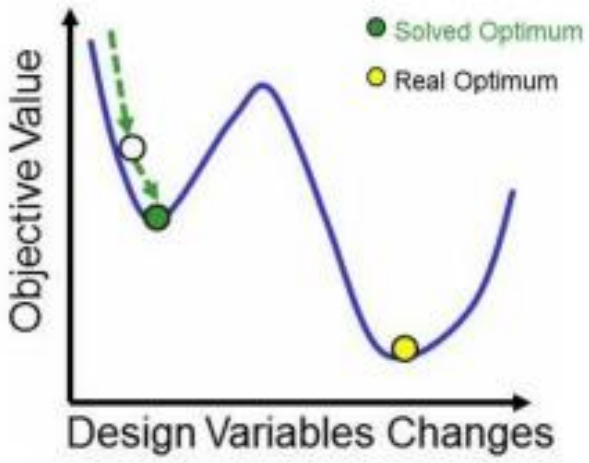
- El aprendizaje, esquema general:



Principal objetivo: aprender *h* (i.e. *g* ~ *h*) usando los datos *D*

Funciones de costo o de pérdida (loss)

- Esta función mide la distancia entre la predicción y el valor objetivo real (*target, label, etiqueta*).
- Durante el entrenamiento de la red neuronal, la función de pérdida es usada para calcular el valor de pérdida.
- El valor de pérdida se correlaciona negativamente con el grado de consistencia o *score* (ej. *accuracy*).
- Cuando entrenamos la red neuronal, necesitamos actualizar los pesos por medio de la minimización del valor de pérdida en el conjunto de entrenamiento.



- Función de pérdida en regresión y clasificación:

$$J(\boldsymbol{w}) = \frac{1}{2N} \sum_{p_i, y_i \in D}^{N} (y_i - \hat{y}_i(\boldsymbol{w}))^2$$

MSE: para problemas de regresión

$$J(\boldsymbol{w}) = -\sum_{p_i, y_i \in D}^{N} y_i \cdot \log \hat{y}_i(\boldsymbol{w})$$

Cross-entropy: para problemas de clasificación

- Este problema de optimización se resuelve usando una solución iterativa basada en gradientes (típicamente).

Funciones de pérdida: algunas visualizaciones

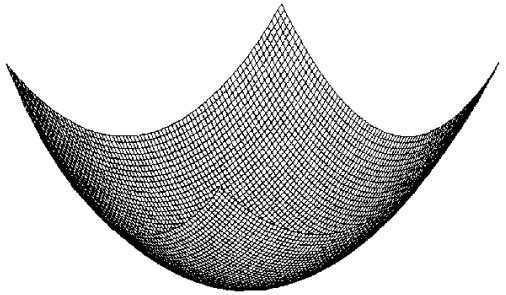


Fig. 22. Example MSE surface of linear error.

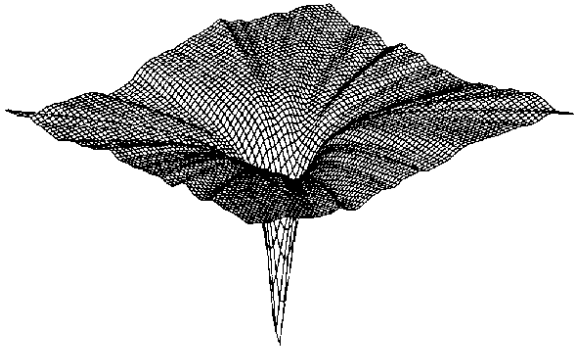


Fig. 23. Example MSE surface of sigmoid error.

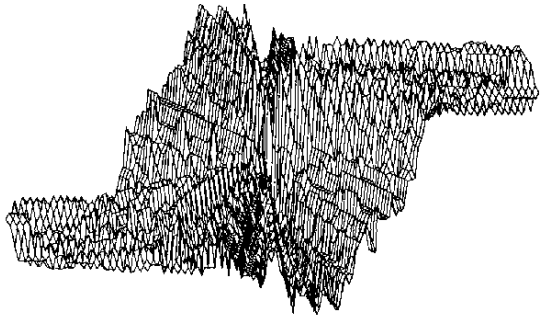
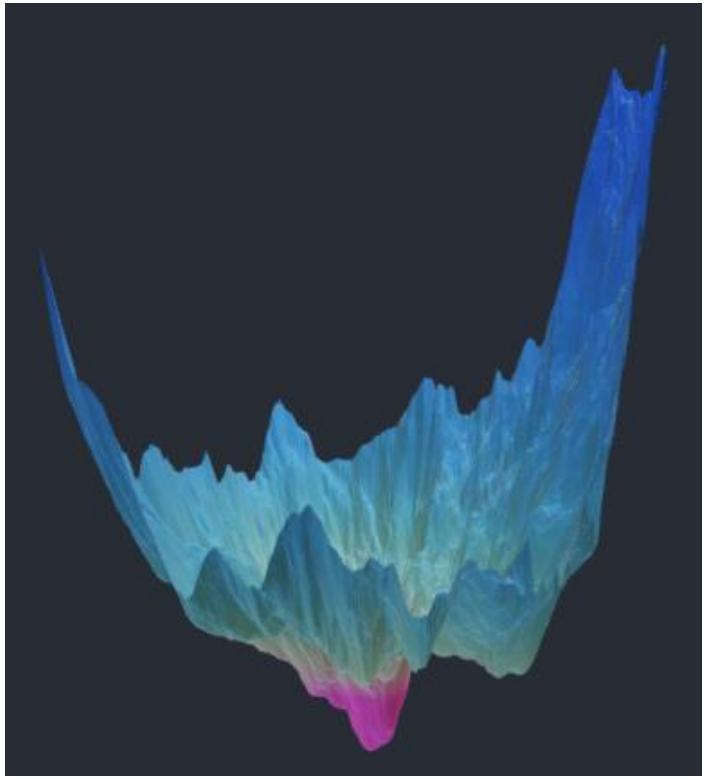
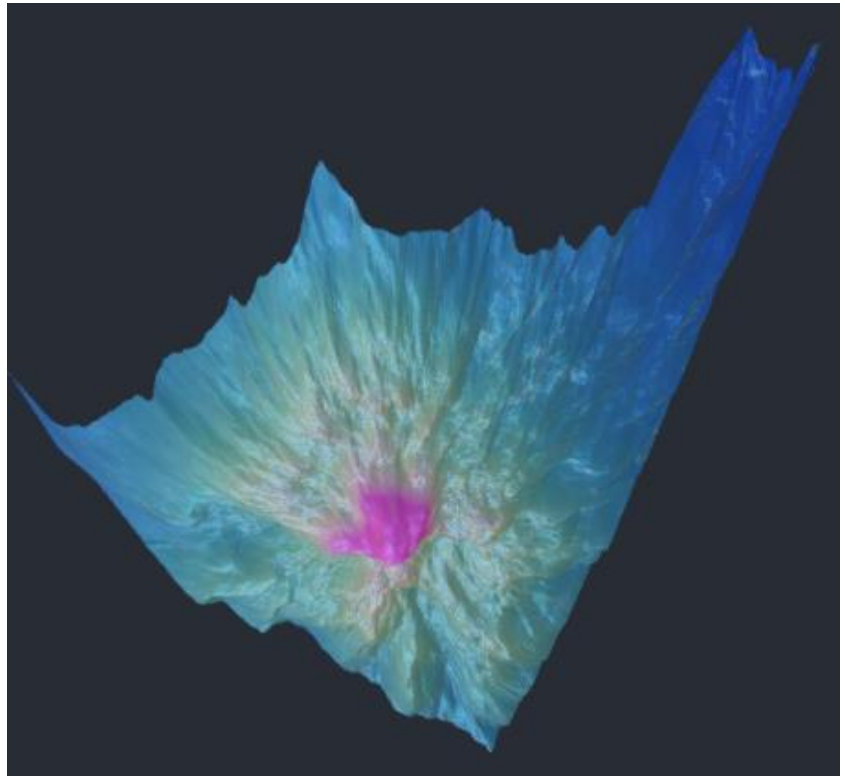


Fig. 24. Example MSE surface of signum error.

fuente





Panorama de la función de pérdida de una red neuronal convolucional sobre el set de datos Imagenette.

https://losslandscape.com/explorer

Algoritmo de Descenso del Gradiente (*Gradient Descent*)

- El **algoritmo de descenso del gradiente** mueve los pesos en la dirección contraria a la del gradiente de forma iterativa, minimizando finalmente la función de pérdida.

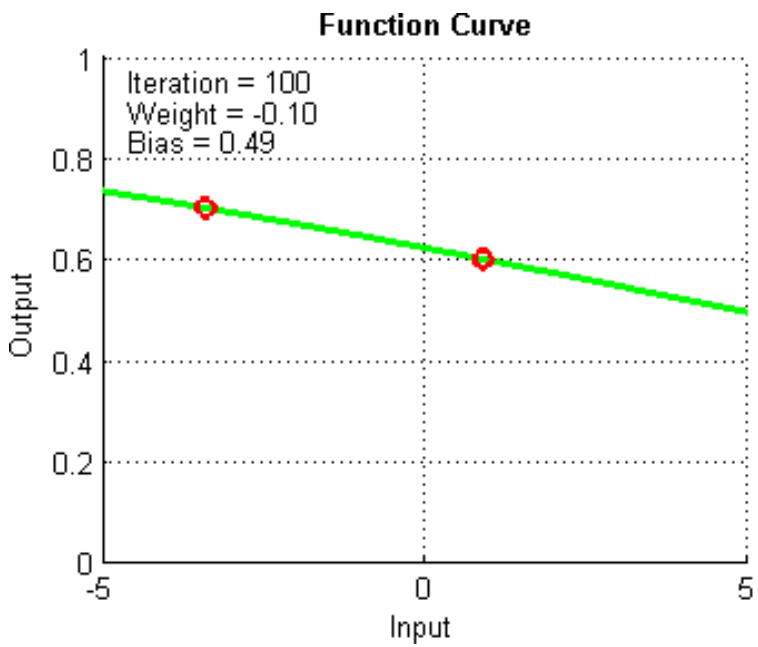
$$w_i \leftarrow w_i + \Delta w_i$$

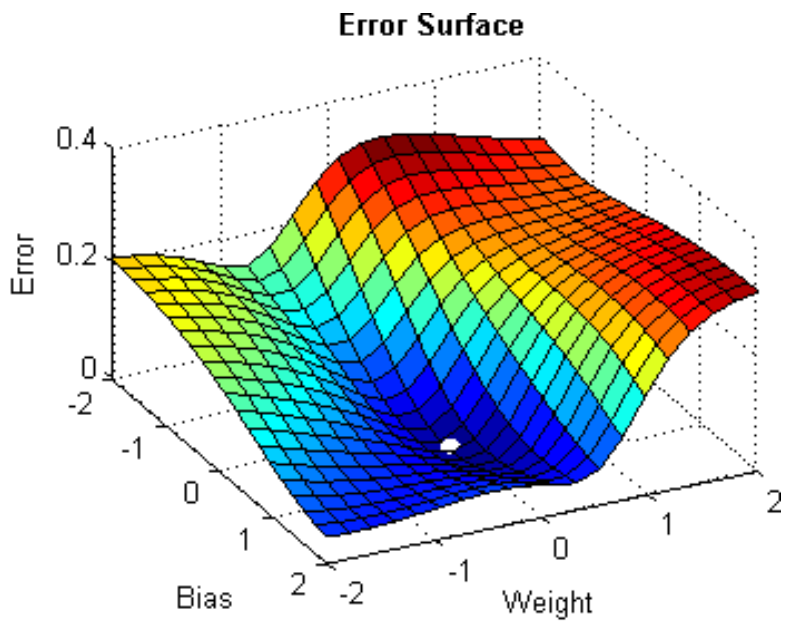
$$\Delta w_i = -\eta \cdot \frac{\partial J}{\partial w_i}$$

$$\Delta \boldsymbol{w} = -\eta \cdot \nabla J(\boldsymbol{w})$$

Tasa de aprendizaje

$$0 < \eta \leq 1$$

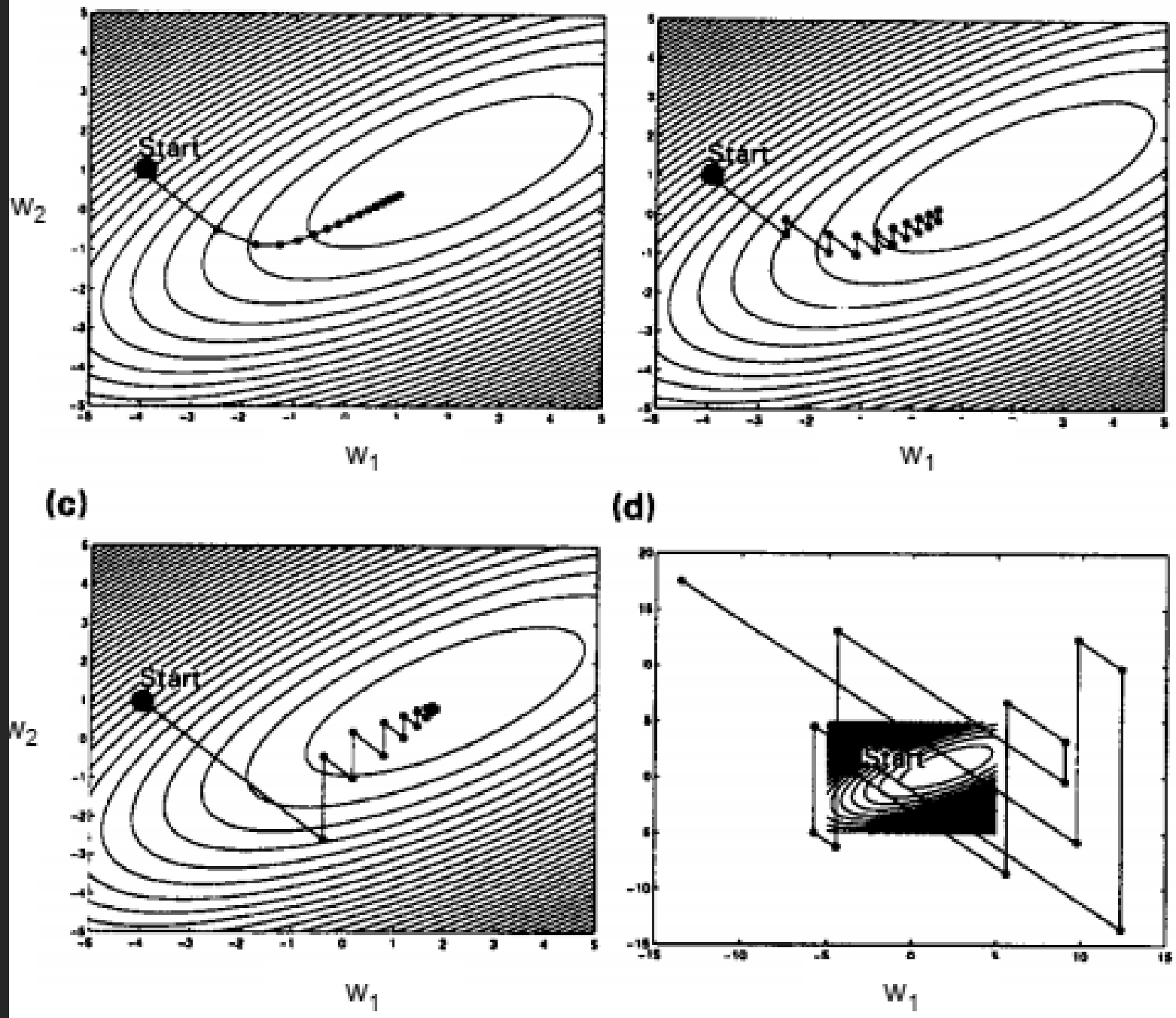




Ejemplo en regresión lineal (fuente)

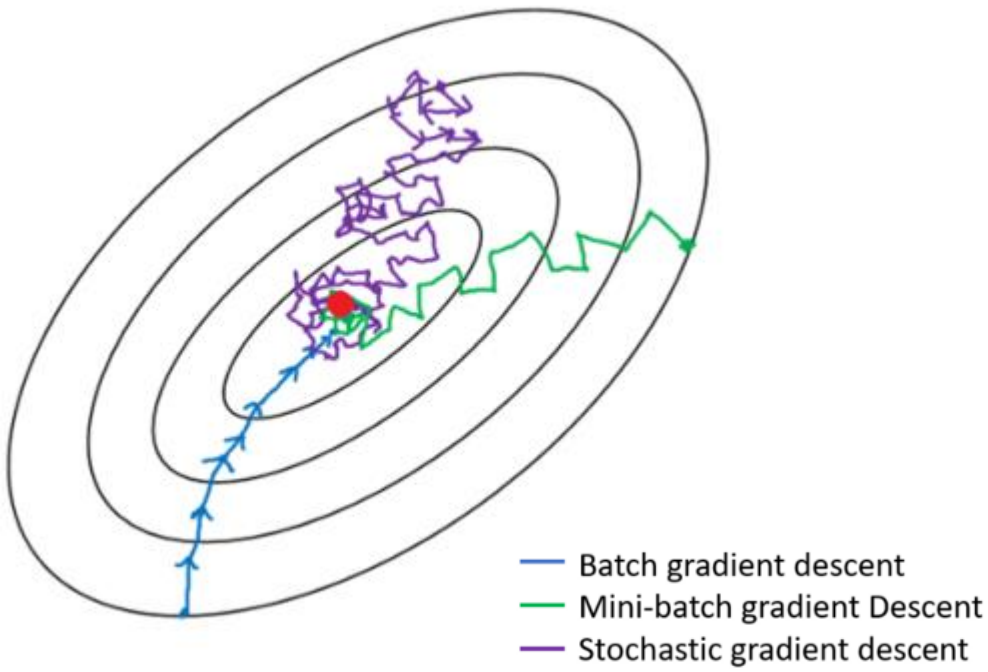
Efecto de la tasa de aprendizaje

(recorder: es un hiperparámetro de la red)



Algoritmo de Descenso del Gradiente: diferentes versiones

- El algoritmo del descenso del gradiente tiene diferentes versiones, dependiendo de la cantidad de datos que se usan en cada iteración.
- Tenemos:
  - Descenso del gradiente estocástico (SGD, *Stochastic Gradient Descent*): En cada iteración actualiza el gradiente basado en muestras individuales.
  - Descenso del gradiente por lotes (BGD, *Batch Gradient Descent*): Utiliza todos los datos de entrenamiento para actualizar el gradiente y pesos en cada iteración.
  - Descenso del gradiente por mini-lotes (MBGD, *Mini-Batch Gradient Descent*): Mezcla los dos anteriores, utiliza porciones más pequeñas del conjunto de datos original para calcular las salidas, gradientes y actualizar los pesos. Es la opción más usada. Los pesos se actualizan solo cuando han pasado todos los lotes.



fuente

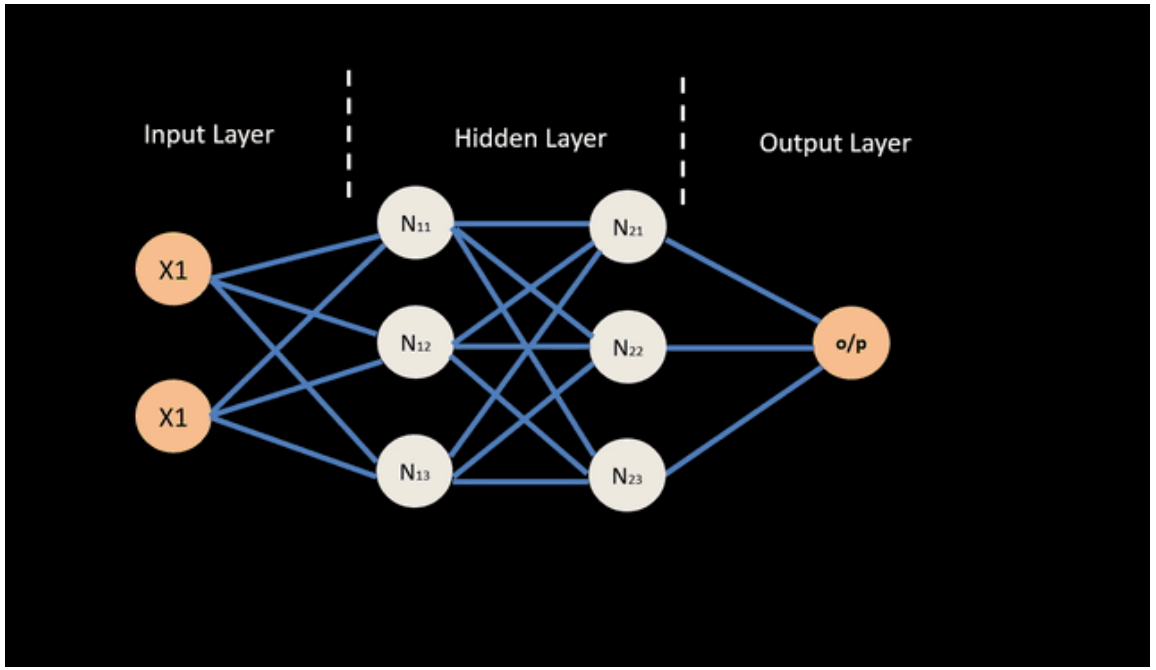
Algoritmo de Propagación hacia atrás (Backpropagation)

- Los algoritmos anteriores son fáciles de implementar para el caso de redes simples de una capa oculta, pero cuando encontramos algunas con muchas capas, el cálculo de gradientes puede complejizarse, aquí aparece el algoritmo de **Propagación hacia Atrás o Backpropagation**, propuesto en 1986 por Rumelhart, Hinton y Williams.

- Los siguientes pasos componen el algoritmo:

  - Previo: Inicialización de los pesos (generalmente de forma aleatoria),
  - propagación hacia delante de los datos (usando los pesos actuales) calculando los errores a la salida,
  - propagación hacia atrás con los errores y gradientes calculados y actualización de los pesos.

- La ventaja consiste en que los gradientes son calculados por cada capa separadamente, esto se logra con la regla de la cadena al tomar las derivadas parciales de la función de pérdida.



fuente

Cómo evitar el sobreajuste

Regularización

La **regularización** es una tecnología importante y efectiva para reducir los errores de generalización en *machine learning*. Es especialmente útil en modelos de *deep learning* que tienden a sobreajustar debido a la gran cantidad de parámetros (capacidad de aprendizaje) que pueden tener. Por consiguiente, los investigadores han propuesto muchas tecnologías efectivas para prevenir el sobreajuste, estas incluyen:

- Añadiendo restricciones a los parámetros, tal como normas L_1 y L_2.
- Expandir el conjunto de datos, por ejemplo, añadiendo ruido y transformando los datos.
- Abandono aleatorio (*Dropout*)
- Detención temprana (*Early Stopping*)

Restricciones a parámetros

- Muchos métodos de regularización restringen la capacidad de los modelos por medio de la adición de parámetros de penalización a la función objetivo J. Asumir que la función objetivo luego de regularización es $\tilde{J}$.

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

- Donde $\alpha \in [0, \infty)$ es un hiperaparámetro que pondera la contribución relativa del término de penalización $\Omega(\theta)$ (típicamente una norma L_1 o L_2) y la función objetivo estándar J. Si α es ajustada a 0, no se aplica ninguna regularización. **La penalización en regularización aumenta con $\boldsymbol{\alpha}$.**

Expansión del conjunto de datos (*Data Augmentation*)

- La forma más efectiva de prevenir el sobreajuste es **sumar datos al conjunto de datos**. Un conjunto de entrenamiento mas grande tiene menor probabilidad de ser sobreajustado. La expansión del conjunto de datos es un método que puede variar en diferentes campos.

- Un método común en el campo de reconocimiento de objetos es rotar o escalar imágenes. El pre-requisito para transformar imágenes es que el tipo de la imagen y sentido semántico de lo que contiene (ej. si hay un perro en la imagen, debe seguir siendo identificado luego de transformar la imagen) no se debe cambiar mediante la transformación.

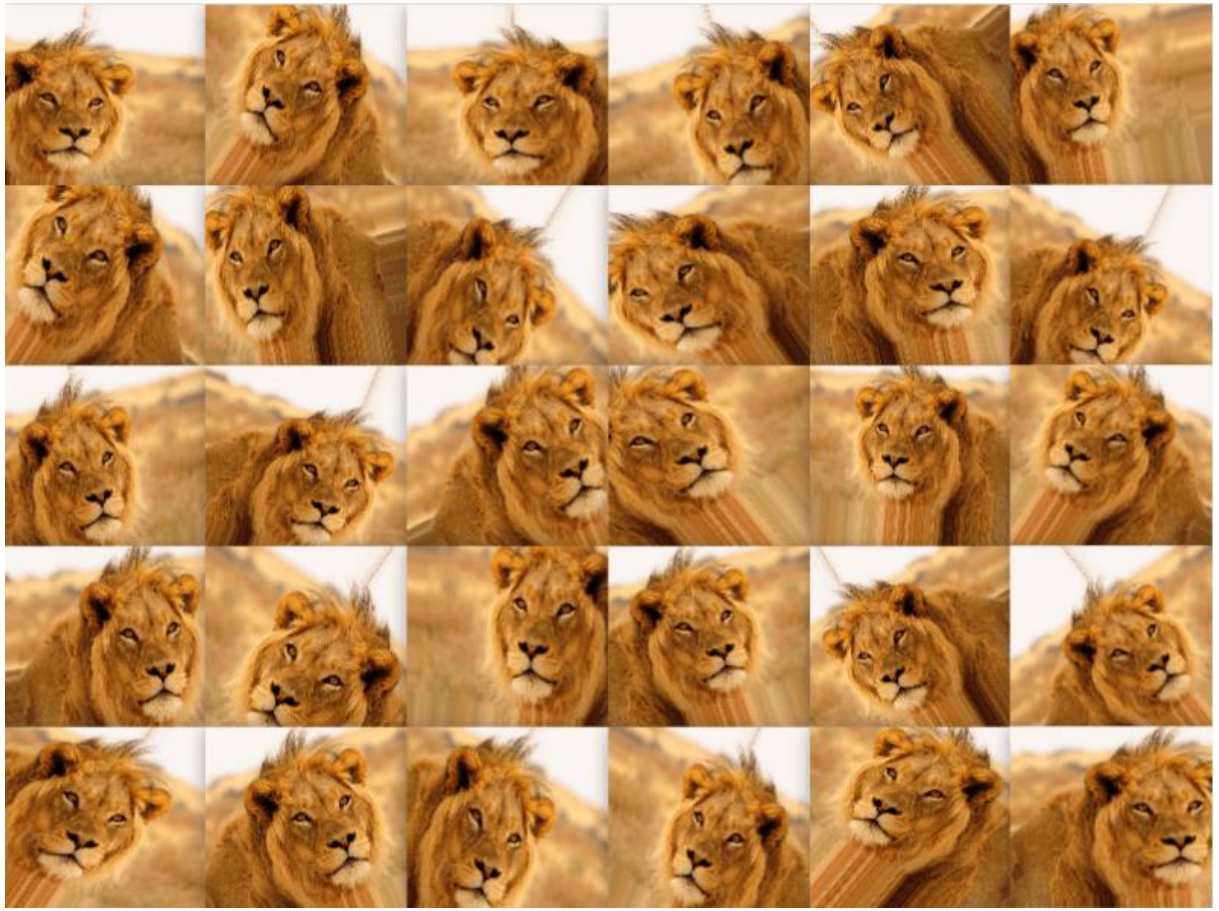
- Adición de ruido aleatorio a datos de entrada en reconocimiento de voz.

- En Procesamiento de Lenguaje Natural (NLP) es común reemplazar palabras con sus sinónimos.

Expansión del conjunto de datos (*Data Augmentation*)

- Ejemplo de expansión de datos con imágenes:

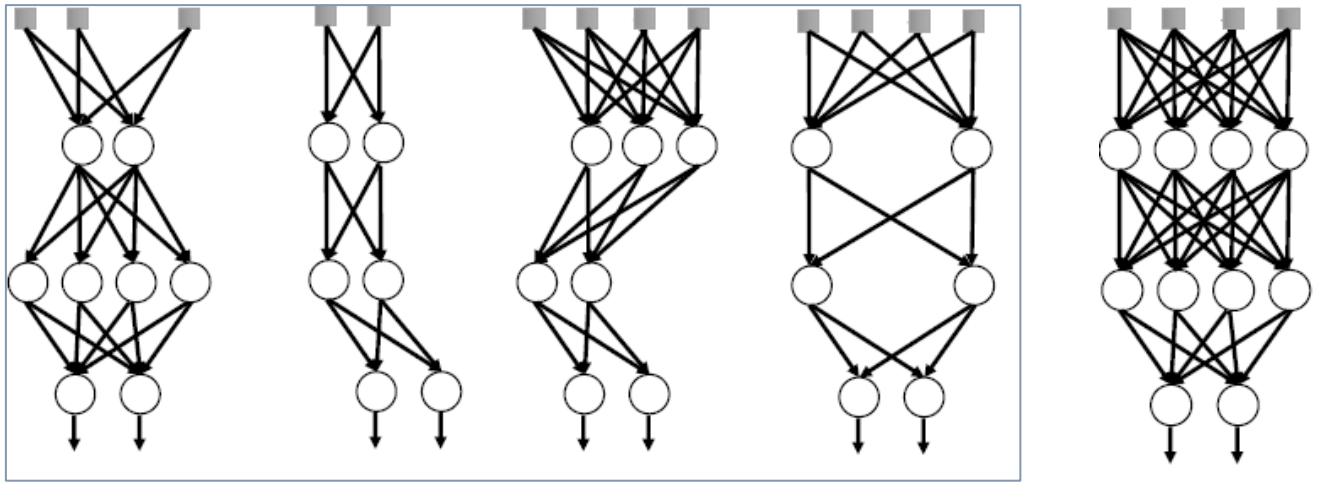
Transformaciones:
- Rotaciones
- Escalado
- Reflexiones
- Recortes
- Vueltas
- Iluminación
- Adición de ruido,
- Etc.



[Fuente]

Abandono aleatorio (Dropout)

- La técnica de Dropout es un método de regularización simple que ha sido usado ampliamente desde el 2014. En palabras simples, el método descarta aleatoriamente algunas entradas a neuronas durante el proceso de entrenamiento y con cierta probabilidad. En este caso, los parámetros correspondiente de las entradas descartadas no son actualizados. Como un método de integración, *dropout* combina todos los resultados de las sub-redes y obtiene sub-redes mediante el descarte o abandono aleatorio de entradas. Ver la figura de abajo:



Dropout in training

Testing

Dropout es menos efectivo cuando el conjunto de datos de entrenamiento es insuficiente.

Dropout: A Simple Way to Prevent Neurals Networks from Overfitting
https://jmlr.org/papers/v15/srivastava14a.html

Abandono aleatorio (Dropout)

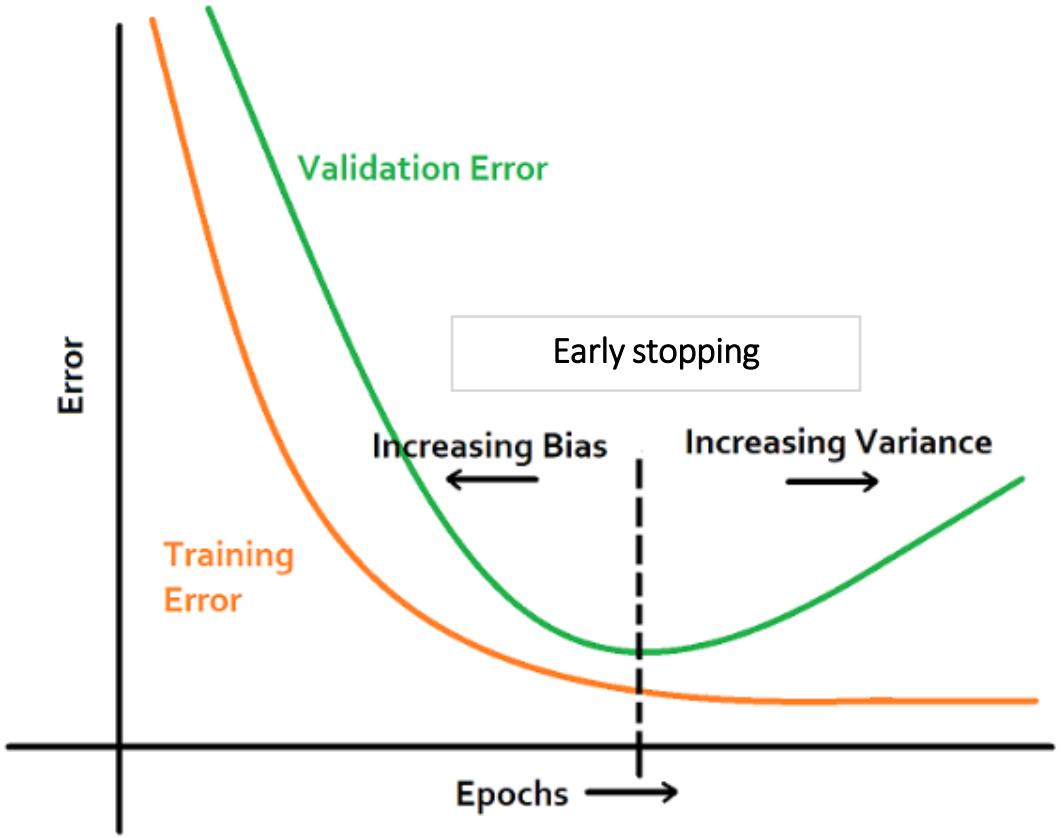
Dropout

Hinton et al.

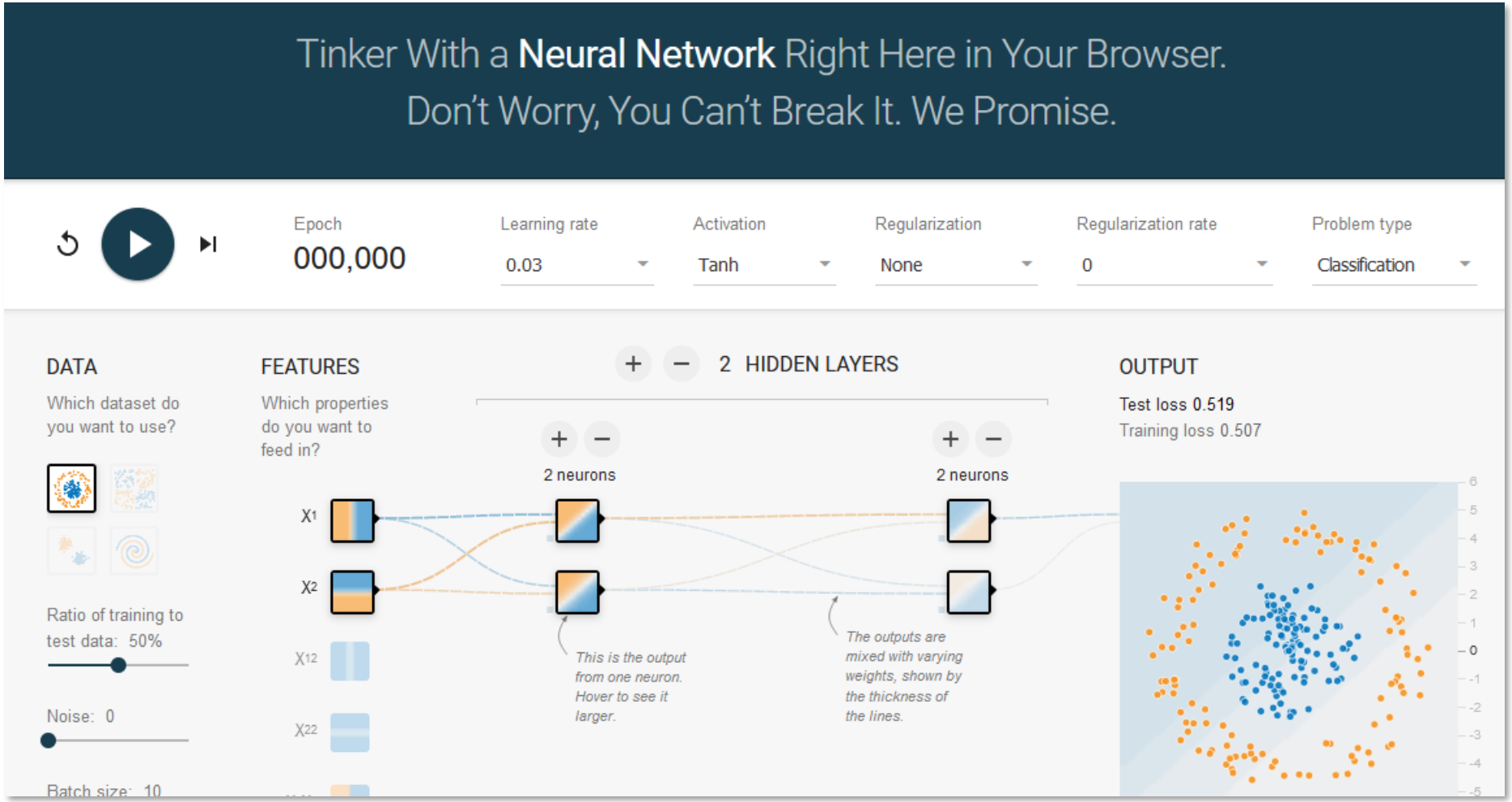
https://pages.cs.wisc.edu/~sharonli/courses/cs540_spring2022/documents/lec14.pdf

Detención temprana (Early Stopping)

- Evaluaciones de prueba en el conjunto de validación pueden ser incluidas durante el entrenamiento. Cuando la función de pérdida aplicada al conjunto de verificación incrementa, se puede implementar detención temprana del proceso.



Jugando con redes neuronales



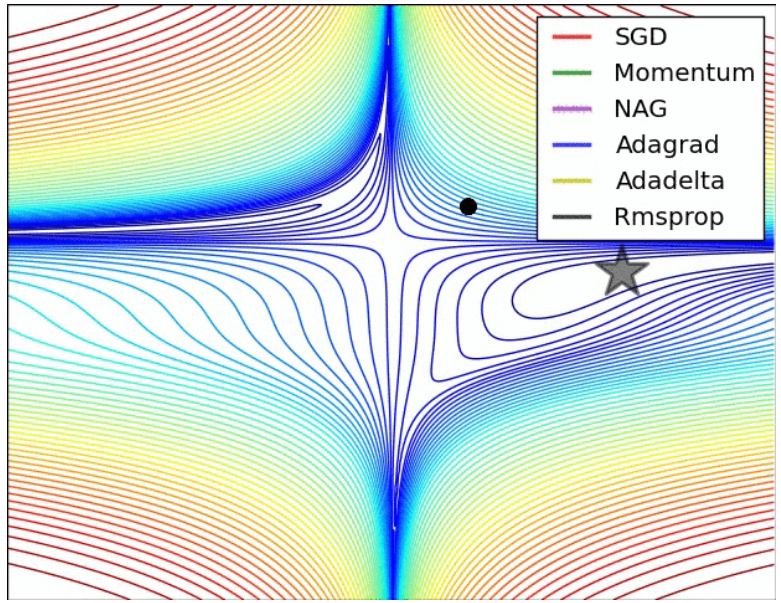
https://playground.tensorflow.org/

Optimizadores

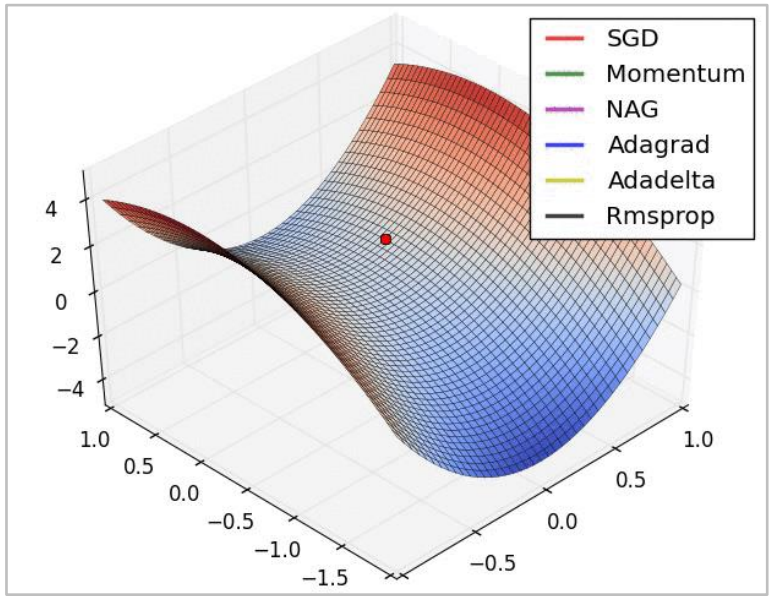
Optimizadores

- Existen varias versiones optimizadas de los algoritmo del descenso del gradiente. En implementaciones con lenguajes orientados a objetos, diferentes algoritmos de descenso del gradiente son a menudo encapsulados en objetos llamados **optimizadores**.
- Los propósitos de los algoritmos de optimización incluyen, aunque no se limitan, a:
  - Acelerar la convergencia del algoritmo
  - Prevenir o saltarse los valores mínimos locales
  - Simplificar el ajuste manual de hiperparámetros, especialmente la tasa de aprendizaje (LR)
- Optimizadores comunes: el clásico optimizador de descenso del gradiente, momentum, Nesterov, AdaGrad, AdaDelta, RMSProp, Adam, AdaMax, y Nadam.

Comparación de desempeño de varios optimizadores



Comparison of optimization algorithms in contour maps of loss functions



Comparison of optimization algorithms at the saddle point

Refs: `https://cs231n.github.io/neural-networks-3/`

Herramientas de programación

Frameworks y herramientas para IA

- Un marco (framework) de aprendizaje profundo es una interfaz, una biblioteca o una herramienta que nos permite construir modelos de aprendizaje profundo más fácilmente y rápidamente, sin entrar en los detalles de los algoritmos subyacentes. Un marco de aprendizaje profundo puede considerarse un conjunto de elementos constitutivos. Cada componente en los bloques de construcción es un modelo o algoritmo. Por lo tanto, los desarrolladores pueden utilizar componentes para ensamblar modelos que cumplan con los requisitos, y no necesitan empezar desde cero.

- La aparición de marcos de aprendizaje profundo reduce los requisitos para los desarrolladores. Los desarrolladores ya no necesitan compilar código a partir de complejas redes neuronales y algoritmos de retropropagacióin. En su lugar, pueden utilizar modelos existentes para configurar los parámetros según se requiera, donde los parámetros del modelo se entrenan automáticamente. Además, pueden agregar capas de red autodefinidas a los modelos existentes, o seleccionar clasificadores y algoritmos de optimización necesarios directamente invocando código existente.

PYTORCH

TensorFlow + Keras

ONNX

Pytorch

- PyTorch es un framework de aprendizaje automático basado en Python desarrollado por Facebook. Es desarrollado sobre la base de Torch, un marco de computación científica apoyado por un gran número de algoritmos de Machine Learning. Torch es una biblioteca de operaciones de tensor similar a NumPy, caracterizada por una gran flexibilidad, pero es menos popular porque utiliza el lenguaje de programación Lua. Por eso PyTorch fue desarrollado.
- Además de Facebook, institutos como Twitter, GMU y Salesforce también utilizan PyTorch.

PYTORCH

https://pytorch.org/

Tensorflow

- TensorFlow es un software de código abierto de Google. Soporta varios algoritmos de aprendizaje profundo y múltiples plataformas de computación, garantizando una alta estabilidad del sistema e integración.

Escalabilidad

GPU, TPU

Computabilidad potente

TensorFlow

https://www.tensorflow.org/

Ejemplo de aplicación 1

Ejemplo de aplicación 2

Ejemplo de aplicación 3

Multi-lenguaje

Multiplataforma

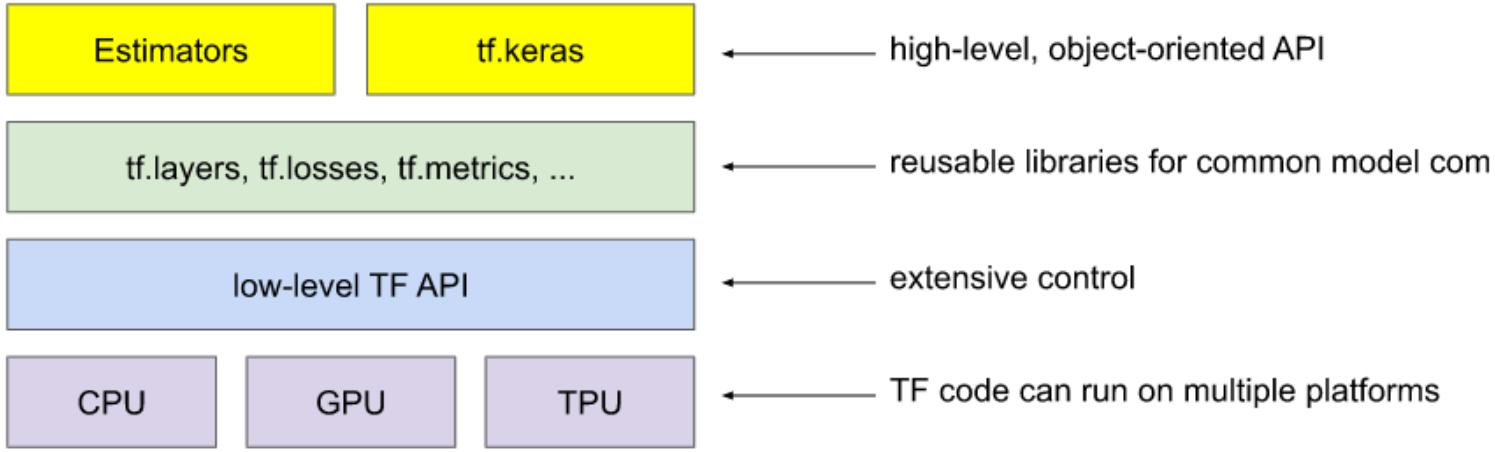
Distribuido

Tensorflow + Keras

- En este curso trabajaremos con la versión 2.x de Tensorflow que incluye una API de alto nivel, Keras, que facilita la programación de los modelos de aprendizaje profundo.

Keras

https://keras.io/



Jerarquía del kit de herramientas de TensorFlow

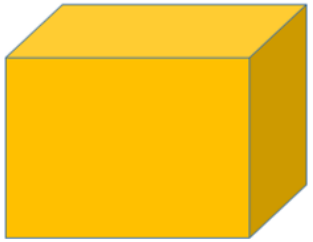
API, *Application Programming Interface*: software intermediario que permite a dos aplicaciones conversar entre ellas.

Tensores

- Los tensores son las estructuras de datos más básicas en TensorFlow. Todos los datos están encapsulados en tensores.
- Tensor: una matriz multidimensional
  - Un escalar es un tensor de rango 0. Un vector es un tensor de rango 1. Una matriz es un tensor de rango 2.
- En TensorFlow, los tensores se clasifican en:
  - Tensores constantes
  - Tensores variables

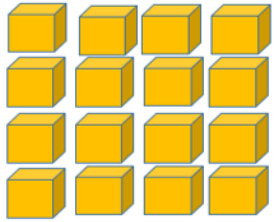
tensor unidimensional

tensor bidimensional

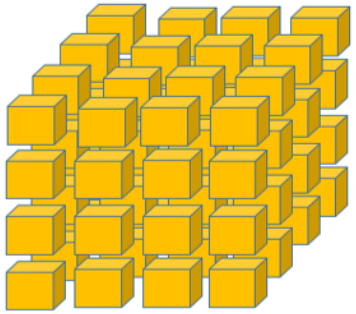


tensor tridimensional

tensor de cuatro dimensiones



tensor de cinco dimensiones



tensor de seis dimensiones

Módulos comúnes de Tensorflow 2.x

- **tf:** Las funciones del módulo tf se utilizan para realizar operaciones aritméticas comunes, como tf.abs (cálculo de un valor absoluto), tf.add (adición de elementos uno por uno) y tf.concat (concatenación de tensores). La mayoría de las operaciones en este módulo pueden ser realizadas por Numpy.
- **tf.errors**: modulo de Tensorflow de tipo error.
- **tf.data**: implementa operaciones en conjuntos de datos.
  - Las tuberías de entrada creadas por tf.data se utilizan para leer datos de entrenamiento. Además, los datos pueden ser fácilmente introducidos desde las memorias (como Numpy).
- **tf.distributions**: implementa varias distribuciones estadísticas.
  - Las funciones de este módulo se utilizan para implementar diversas distribuciones estadísticas, como la distribución Bernoulli, la distribución uniforme y la distribución gaussiana.

Módulos comúnes de Tensorflow 2.x

- **tf.io.gfile:** implementa operaciones en archivos.
  - Las funciones de este módulo se pueden utilizar para realizar operaciones de archivo I/O, copiar archivos y renombrar archivos.
- **tf.image:** implementa operaciones en imágenes.
  - Las funciones de este módulo incluyen funciones de procesamiento de imágenes. Este módulo es similar a OpenCV, y proporciona funciones relacionadas con la luminancia de la imagen, la saturación, la inversión de fase, el recorte, el cambio de tamaño, la conversión de formato de imagen (de RGB a HSV, YUV, YIQ o gris), rotación, y detección de bordes de sobel. Este módulo es equivalente a un pequeño paquete de procesamiento de imágenes de OpenCV.
- **tf.keras**: una API de Python para invocar las herramientas de Keras.
  - Se trata de un módulo grande que permite varias operaciones de red.

Interfaz Keras

- TensorFlow 2.x recomienda Keras para la construcción de redes. Las redes neuronales comunes se incluyen en **Keras.layers**.
- Keras es una aplicación de alto nivel que se utiliza para construir y entrenar modelos de aprendizaje profundo. Se puede utilizar para el diseño rápido de prototipos, la investigación avanzada y la producción. Tiene las siguientes tres ventajas :
  - Fácil de usar
    - Keras proporciona GUI simples y consistentes optimizadas para casos comunes. Proporciona información práctica y clara sobre los errores del usuario.
  - Modular y componente
    - Se pueden crear modelos con Keras conectando bloques de construcción configurables entre sí, con pocas restricciones.
  - Fácil de extender
    - Se pueden personalizar los bloques de construcción para expresar nuevas ideas de investigación, crear capas y funciones de pérdida, y desarrollar modelos avanzados.

Métodos e interfaces comunes de Keras

- A continuación se describen los métodos e interfaces comunes de tf.keras centrándose en el código. El contenido principal es el siguiente:
  - Procesamiento de conjuntos de datos: conjuntos de datos y preprocesamiento
  - Creación de modelos de red neuronal: secuencial (Sequential), modelo (Model), capas (Layers)...
  - Compilación de red: Compilar (Compile), Pérdidas (Losses), Métricas (Metrics) y Optimizadores (Optimizers)
  - Entrenamiento y evaluación de redes: ajuste (fit), ajuste_generador (fit_generator) y evaluación (evaluate)

Recursos extra de aprendizaje

- ❑ Documentación oficial de Tensorflow: enlace
- ❑ Canal de youtube de Tensorflow: enlace
- ❑ Tensorflow Datasets: enlace
- ❑ Tensorflow Hub con modelos de Deep learning pre-entrenados: enlace
- ❑ Curso de Deep learning de Google Developers: enlace
- ❑ Especializaciones en Deep Learning de acceso gratuito (se paga solo el certificado): enlace
- ❑ Un curso introductorio de Machine Learning de Google developers (videos disponibles en español): enlace
- ❑ Curso Tensorflow, Keras and Deep Learning, without a PhD: enlace

Referencias

Parte del material de este curso esta basado en:

- Certificación Huawei HCIA-AI
- Documentación oficial Tensorflow y Keras.
- Curso TensorFlow, Keras and Deep Learning, without a PhD:

  `https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist#0`

Gracias

fuente