

Repaso de redes neuronales artificiales

Métodos Generativos, curso 2025-2026

Guillermo Iglesias, guillermo.iglesias@upm.es

Jorge Dueñas Lerín, jorge.duenas.lerin@upm.es

Edgar Talavera Muñoz, e.talavera@upm.es

5 de noviembre de 2025

Escuela Técnica Superior de Ingeniería de Sistemas Informáticos | UPM

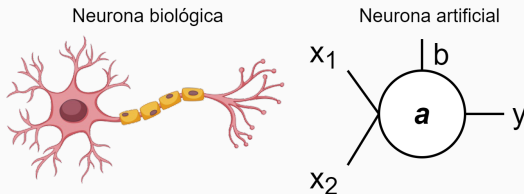


Repaso de conceptos básicos de redes neuronales

Redes neuronales: concepto general

Una **red neuronal** es un sistema matemático capaz de aprender a realizar predicciones a partir de unos datos de entrada, fue propuesta por McCulloch y Pitts en 1943[1], se basaba en **imitar** el comportamiento de una neurona biológica.

- Toma ciertos **estímulos** de entrada, los procesa y genera una nueva salida.
- En el caso biológico los estímulos provienen de **impulsos nerviosos**, mientras que en el caso de las neuronas artificiales esto es replicado a través de cálculos matemáticos.



Grandes hitos en redes neuronales

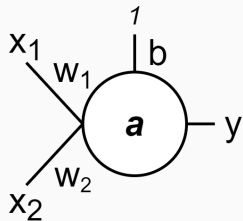
Las redes neuronales han evolucionado a través de hitos clave que han marcado su desarrollo:

- **1943: Neurona artificial** (McCulloch y Pitts): Primer modelo matemático inspirado en la biología
- **1958: Perceptrón** (Rosenblatt): Uso de funciones de activación como el umbral
- **1986: Retropropagación** (Rumelhart et al.): Entrenamiento efectivo de redes profundas
- **1989: CNNs** (LeCun et al.): Redes convolucionales para visión por computador
- **Inviernos de la IA:** Estancamiento. Límites técnicos/expectativas no cumplidas
- **2012: AlexNet:** **GPUs** y big data impulsan el aprendizaje profundo moderno
- **2017: Transformers** (Vaswani et al.): Nueva arquitectura para NLP.
- **2020s: Difusión:** Modelos generativos para imágenes y datos sintéticos
- **Actualidad:** Grandes modelos multimodales
- **Futuro:** ¿Invierno o Inteligencia artificial general?

Redes neuronales: concepto general

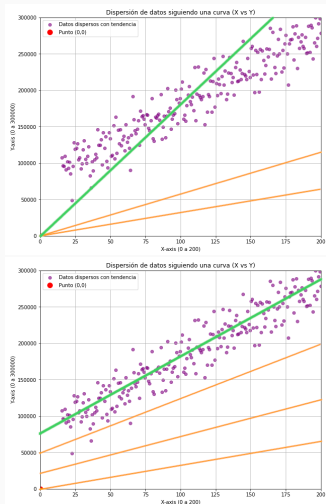
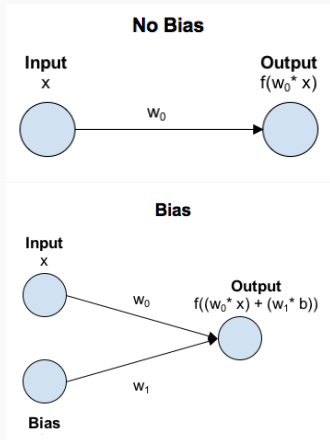
Una **neurona artificial** realiza cálculos matemáticos para transformar ciertos valores numéricos. Para dicha labor, existen diversos **elementos** dentro de una neurona artificial:

- **Entradas** (x_i): Introducen valores numéricos en la neurona
- **Salida** (y): Suministra la salida de la neurona
- **Pesos** (w_i): Son parámetros capaces de cambiar, realizan el aprendizaje de la neurona
- **Bias** (b): Realiza la misma función que cualquier otro peso, pero el valor de su entrada es **siempre** 1
- **Función de activación** (a): Es una función que participa en el cálculo de la salida de la neurona



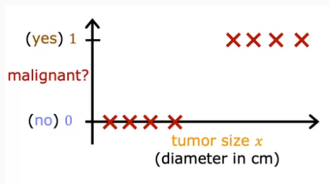
Redes neuronales: bias

El **bias** (sesgo en Español) permite que la red neuronal pueda ajustar mejor sus predicciones.

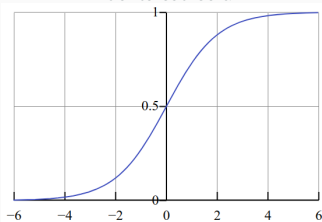


Redes neuronales: activación

Capas de activación. En salida.



Fuente Coursera



Wikipedia

Sigmoid

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

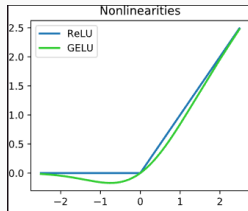
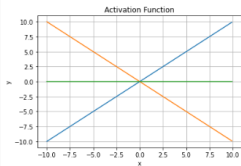
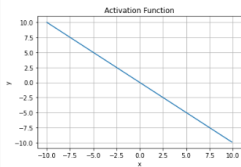
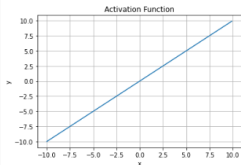
Ejemplo

$$y = \frac{1}{1 + e^{-(3*x-2)}} \quad (2)$$

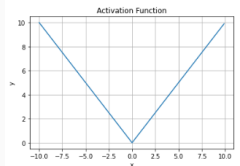
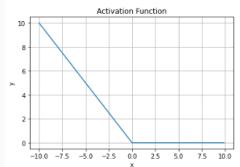
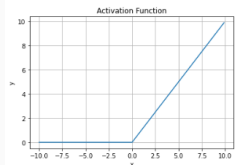
Calculadora gráfica

Redes neuronales: activación

Capas de activación. No linealidad.

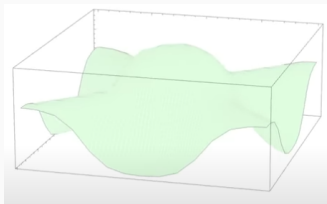
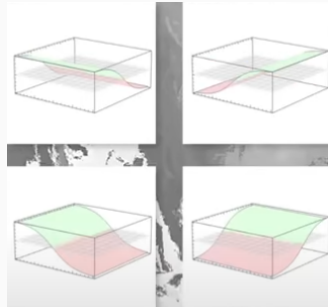
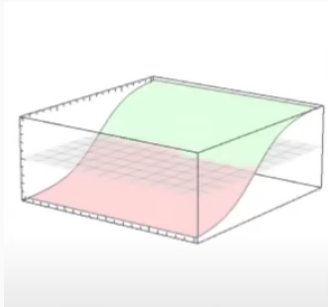


Wikipedia



Redes neuronales: activación

Capas de activación. No linealidad. 3D



DotCSV

Redes neuronales: entrenamiento

El entrenamiento de las redes neuronales se puede dividir en tres fases:

- **Predicción:** calculamos para las entradas la predicción de valor de salida de la red.
- **Calculo de error:** con la predicción y el resultado que queremos obtener calculamos el error obtenido.
- **Ajuste de pesos:** con el error se reajustan los parámetros de la red.

Interviewer: What's your biggest strength?

Me: Machine Learning.

Interviewer: What's $6 + 9$?

Me: 0.

Interviewer: Incorrect. It's 15.

Me: It's 15.

Interviewer: What's $4 + 20$?

Me: It's 15.

Algoritmo de propagación

La fase de **predicción** de una red neuronal se realiza a través del algoritmo de **propagación**.

Este se encarga de procesar la **entrada** y generar la **salida** correspondiente.

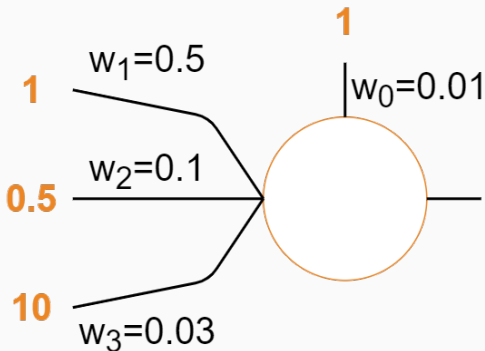
Para ello, la computación de cada **neurona** es la siguiente:

- Cada entrada x_i es **multiplicada** por el valor de su peso correspondiente w_i .
- La entrada de *bias* **siempre** es “1”, y se multiplica por su peso correspondiente (a veces indicado como w_0).
- Todas las **entradas** de la neurona se combinan haciendo una **suma** de todas ellas, de tal manera que se realiza una **combinación lineal**.
- El resultado de la combinación lineal se pasa por una **función no lineal** para generar la **salida** de la neurona.

Algoritmo de propagación

La ecuación que define este **proceso** es la siguiente:

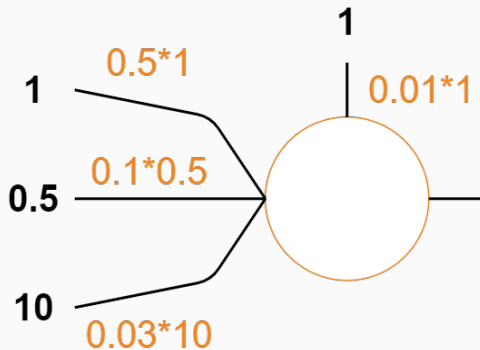
$$f\left(\sum_{i=0}^n w_i X_i\right) \quad (1)$$



Algoritmo de propagación

La ecuación que define este **proceso** es la siguiente:

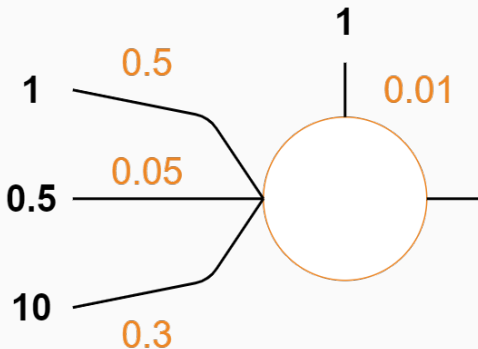
$$f\left(\sum_{i=0}^n W_i X_i\right) \quad (1)$$



Algoritmo de propagación

La ecuación que define este **proceso** es la siguiente:

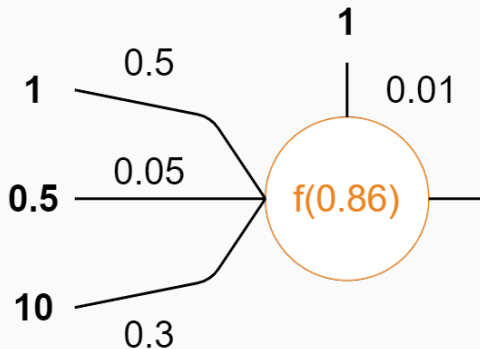
$$f\left(\sum_{i=0}^n W_i X_i\right) \quad (1)$$



Algoritmo de propagación

La ecuación que define este **proceso** es la siguiente:

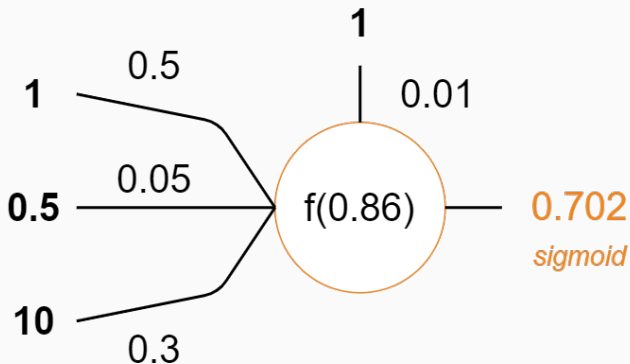
$$f\left(\sum_{i=0}^n W_i X_i\right) \quad (1)$$



Algoritmo de propagación

La ecuación que define este **proceso** es la siguiente:

$$f\left(\sum_{i=0}^n W_i X_i\right) \quad (1)$$



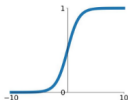
Funciones de activación

Las **funciones de activación** de cada neurona pueden variar, entre las más populares se encuentran:

Activation Functions

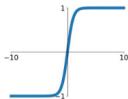
Sigmoid

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



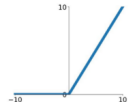
tanh

$$\tanh(x)$$



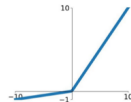
ReLU

$$\max(0, x)$$



Leaky ReLU

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

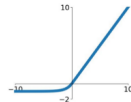


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

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

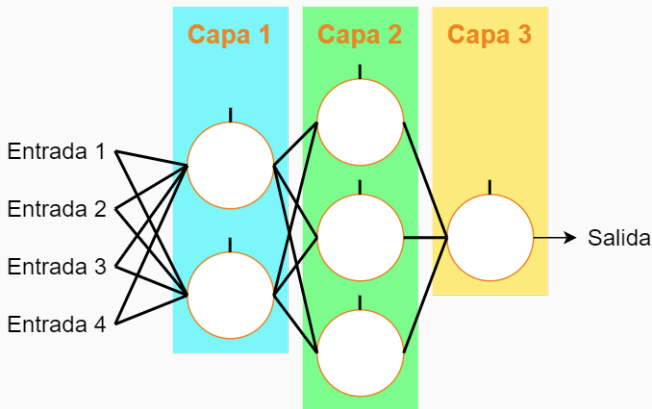


[2]

Estructura de capas

Una red de neuronas “estándar” se organiza por **capas**, las cuales se componen por varias **neuronas**.

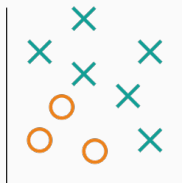
Cada **capa de neuronas** se conecta con la siguiente y recibe **datos** de la anterior. De esta manera se produce el **flujo de datos** a lo largo de la red.



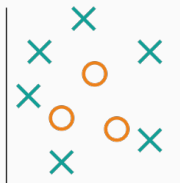
¿Por qué introducir más capas?

Está matemáticamente **demostrado** que sin función de activación las redes de neuronas sólo son capaces de resolver problemas **linealmente separables**.

Esto es fácilmente demostrable, ya la computación de cada neurona corresponde con la ecuación de **una recta**, y su combinación también.



Linealmente separable



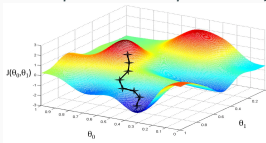
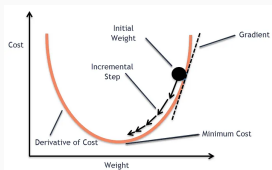
Linealmente no separable

Por otra parte, K. Hornik, M. Stinchcombe, y H. White demostraron el 1985 que con **una única capa oculta** las redes neuronales artificiales se convierten en aproximadores universales [3].

Gradient descent y learning rate

Descenso de gradiente: algoritmo de optimización iterativo de primer orden que permite encontrar mínimos **locales** en una función diferenciable. (Wikipedia).

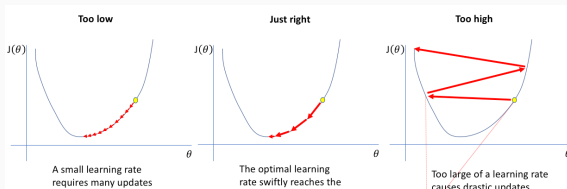
¿Cómo funciona? Derivadas parciales para bajar la pendiente.



¡Ojo! pérdida no datos

Gradient descent un parámetro (learning rate)

Gradient descent w y b(batch size) y Optimizadores



Algoritmo de retropropagación

El algoritmo de **retropropagación** o **backpropagation** es el encargado de **adaptar** la red de neuronas a su cometido específico.

Se basa en actualizar los **pesos** de la red dependiendo del **error** que esta haya tenido a la hora de predecir una **salida** en concreto.

Con backpropagation obtendremos los **gradientes (derivadas)** de la función de pérdida para cada **peso** de cada **capa oculta**.

$$\Delta w_i = w_i - \alpha \cdot (Error) \quad (2)$$

donde α es el **learning rate**, que define la **magnitud** con la que la red realiza la **actualización** de sus pesos.

Backpropagation interactivo

Funciones de pérdida

Existen múltiples métodos para calcular la **distancia** de la **predicción** \hat{y} con respecto de la **salida deseada** y . Es decir, múltiples funciones de pérdida que nos permiten calcular el error.

$$\text{MAE} = \frac{1}{n} \sum |y - \hat{y}| \quad (3)$$

$$\text{MSE} = \frac{1}{n} \sum (y - \hat{y})^2 \quad (4)$$

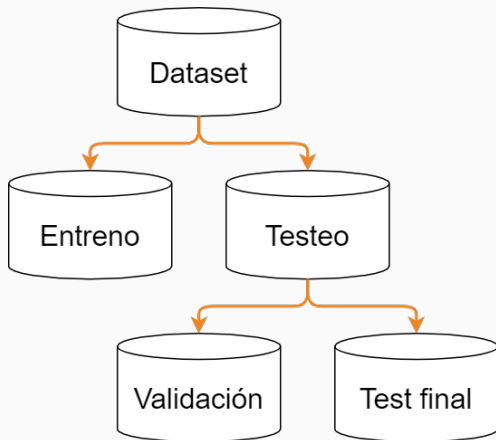
$$\text{Binary Cross-Entropy} = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})) \quad (5)$$

Dibujo de la función **log**. Condicional if $\rightarrow y * \dots + (1 - y) * \dots$

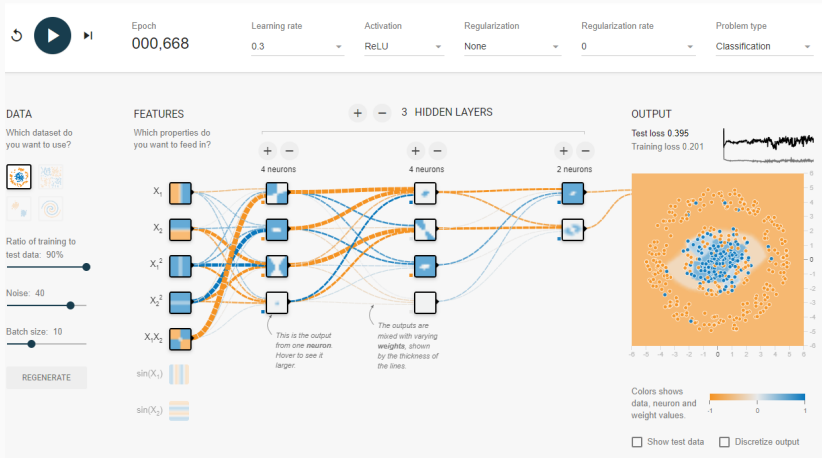
$$\text{Cross-Entropy} = - \sum_{i=1}^N y \cdot \log \hat{y} \quad (6)$$

Entrenamiento de redes neuronales

Al realizar un entrenamiento con **modelos de aprendizaje** se realiza una división del **conjunto de datos** con el que se entrena. Este proceso ayuda a comprobar la **fiabilidad** de la red.



Playground



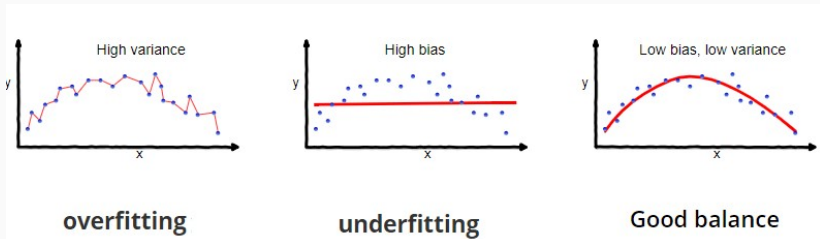
Tensorflow playground

Ejercicios 1 y 2 completo. Ejercicio 3 introducción a siguientes conceptos.

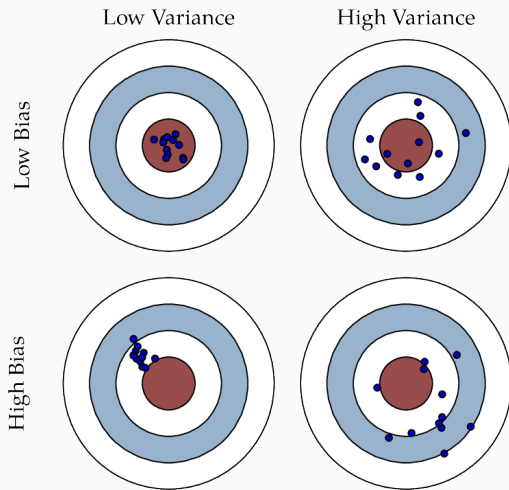
Bias-variance tradeoff

Existen dos **conceptos fundamentales** que ayudan a analizar el rendimiento de una red neuronal:

- **Bias (sesgo)**: Error sistemático que aparece cuando el modelo es demasiado simple para capturar la complejidad de los datos. Se manifiesta en un alto error incluso en el conjunto de **entrenamiento**.
- **Variance (varianza)**: Error debido a la excesiva sensibilidad del modelo a los datos de entrenamiento. Se observa cuando existe una gran diferencia entre el error en **entrenamiento** y el error en **test**.



Bias-variance tradeoff



[5]

¿Cómo detectar alto bias o variance?

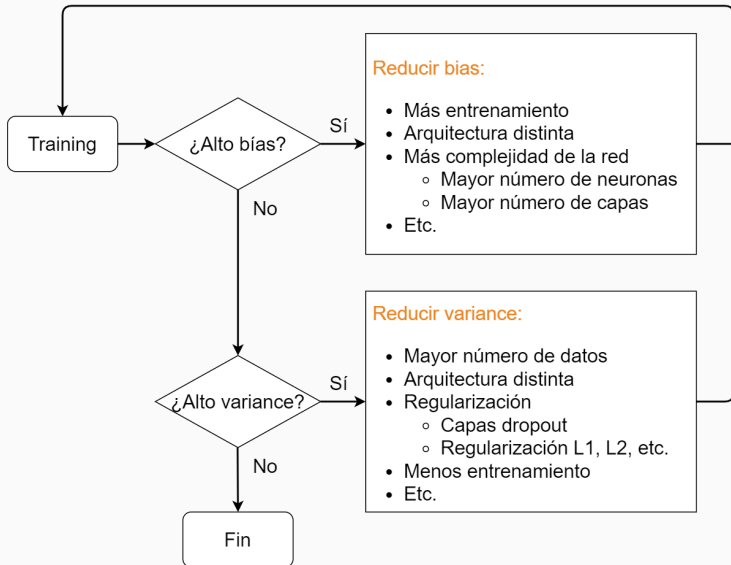
Alto bias

- Underfitting.
- Sobre-simplificación del problema.
- Valores de pérdida demasiado altos.
- Falla al capturar la tendencia de los datos.

Alto variance

- Overfitting.
- Dataset demasiado ruidoso.
- Demasiada complejidad.

Esquema general de entrenamiento de redes neuronales



Problemas del gradiente

Los problemas **derivados del gradiente** son comunes a todas las redes neuronales. Estos están **directamente influenciados** por el número de capas de la red.

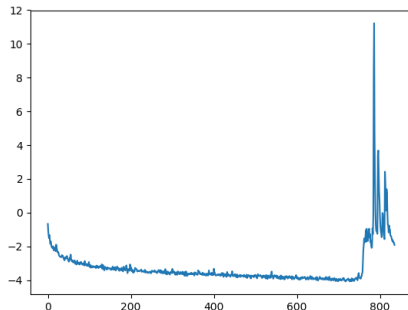
Se diferencian dos tipos:

- Gradient explosion.
- Gradient vanishing.

Al realizarse la **retropropagación** los **valores de pérdida** pasan de unas capas a otras. En este algoritmo las derivadas de cada neurona pueden llegar a **descontrolarse**.

$$W'_x = W_x - \alpha \left(\frac{\partial \text{Loss}}{\partial W_x} \right) \quad (7)$$

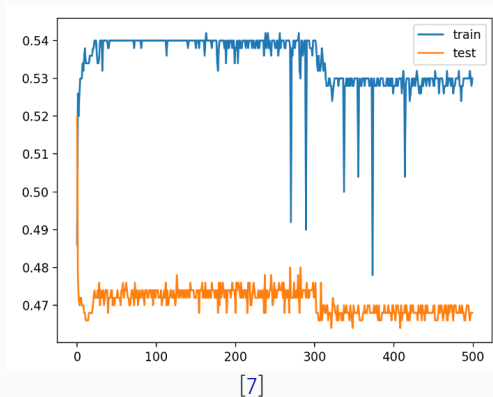
Problemas del gradiente: Gradient explosion



[6]

El **gradient explosion**, también conocido como **exploding gradients** sucede cuando la actualización de pesos toma valores **muy elevados**. Se identifica con valores de pérdidas de **NaN** o **muy exageradas**.

Problemas del gradiente: Gradient Vanishing

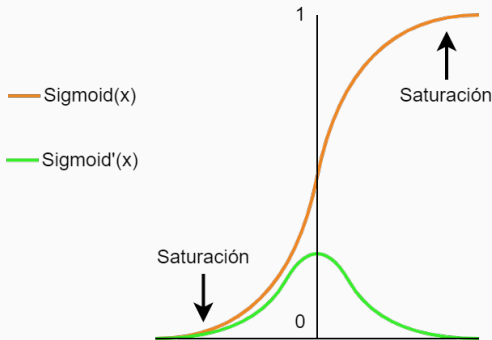


Cuando sucede **gradient vanishing**, también llamado **vanishing gradients**, la actualización de pesos se hace **nula** por tener valores **muy pequeños**.

Se identifica cuando la pérdida es **constante en el tiempo**.

Origen de los problemas derivados del gradiente

La principal **causa** de estos problemas es usar **funciones de activación** cuya derivada **satura a 0**.



Sucede principalmente con las funciones **tanh** y **sigmoid**, por lo tanto se **recomienda** el uso de ReLU para capas ocultas en una red.

Notebook de ejemplo, perceptrón clasificador

El siguiente notebook contiene un ejemplo de clasificador usando un perceptrón multicapa como red neuronal.

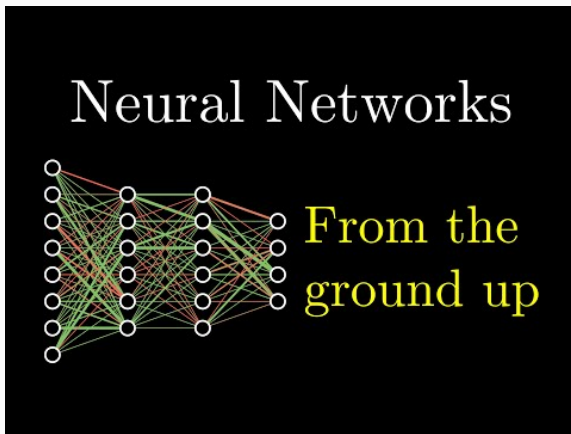


- [1.1_1-PerceptronClasificador.ipynb](#)

Perceptrón multicapa para procesar imágenes

¿Cómo procesar imágenes?

La idea más **básica** para procesar imágenes con redes neuronales es transformar la **matriz numérica** de datos a un **vector unidimensional**.



[Vídeo youtube](#)

La capa de **keras** llamada “**reshape**” se encarga de realizar transformaciones en los tensores. Ya que solo queremos convertir la imagen en un vector “plano”, también podemos utilizar la capa “**flatten**” de **matriz** a **vector**.

Sin embargo el principal **inconveniente** al tratar las imágenes de esta manera es la **pérdida total** de información espacial de la imagen.

Notebook de ejemplo, perceptrón clasificador de imágenes

El siguiente notebook contiene un código parcial de clasificador de imágenes usando un perceptrón multicapa como red neuronal.



- [1.1_2-PerceptronImagenes.ipynb](#)

Si quieres usar estas redes sencillas en otros datasets puede probar con [Wine dataset](#) y con [Fashion MNIST](#).

- [1] Warren S McCulloch and Walter Pitts.
A logical calculus of the ideas immanent in nervous activity.
The bulletin of mathematical biophysics, 5(4):115–133, 1943.
- [2] Data Science Interview Preparation.
Activation functions image.
<https://www.datasciencepreparation.com/blog/articles/what-is-an-activation-function-what-are-commonly-used-activation-functions/>.
[Online; accessed August, 2022].
- [3] Kurt Hornik, Maxwell Stinchcombe, and Halbert White.
Multilayer feedforward networks are universal approximators.
Neural networks, 2(5):359–366, 1989.

- [4] The Machine Learners.
Bias variance tradeoff image.
<https://www.themachinelearners.com/tradeoff-bias-variance/>.
[Online; accessed August, 2022].
- [5] endtoend.ai.
Bias variance tradeoff image.
<https://www.endtoend.ai/blog/bias-variance-tradeoff-in-reinforcement-learning/>.
[Online; accessed August, 2022].
- [6] acrosson (GitHub).
Gradient explosion image.
<https://github.com/NVIDIA/waveglow/issues/122>.
[Online; accessed August, 2022].

[7] Jason Brownlee (Machine Learning Mastery).

Gradient vanishing image.

<https://machinelearningmastery.com/how-to-fix-vanishing-gradients-using-the-rectified-linear-activation-function/>.

[Online; accessed August, 2022].

- Autor original de las diapositivas: Guillermo Iglesias Hernández
- Extensión de contenido: Jorge Dueñas Lerín