clusterAI 2020
ciencia de datos en ingeniería industrial
UTN BA
curso I5521

clase_10: intro a redes neuronales

docente: martin palazzo

redes neuronales

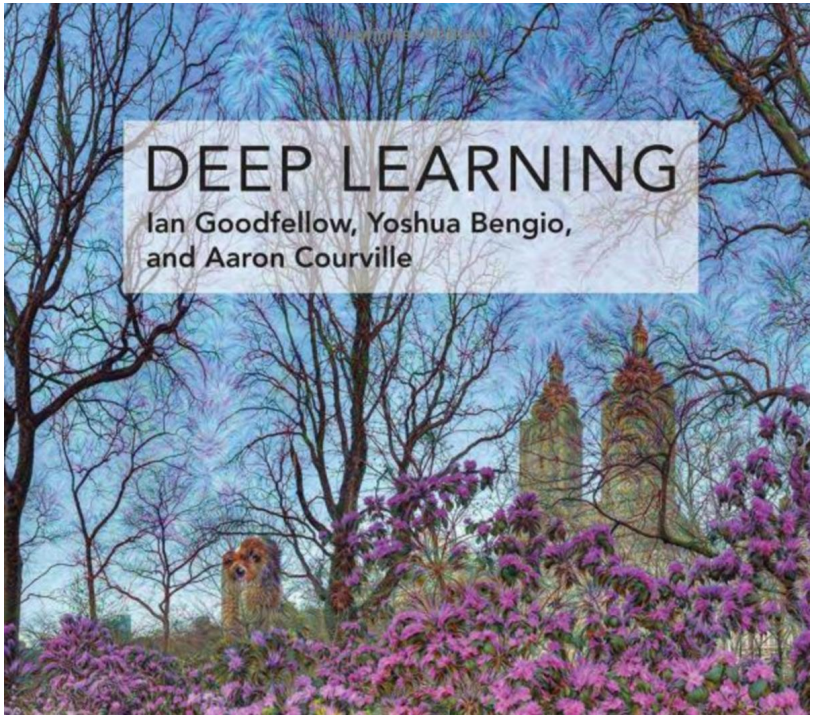
- perceptrón
- funciones de activación
- multilayer perceptrón
- arquitecturas de redes neuronales
- Entrenamiento de una red neuronal

¿que es una red neuronal artificial?

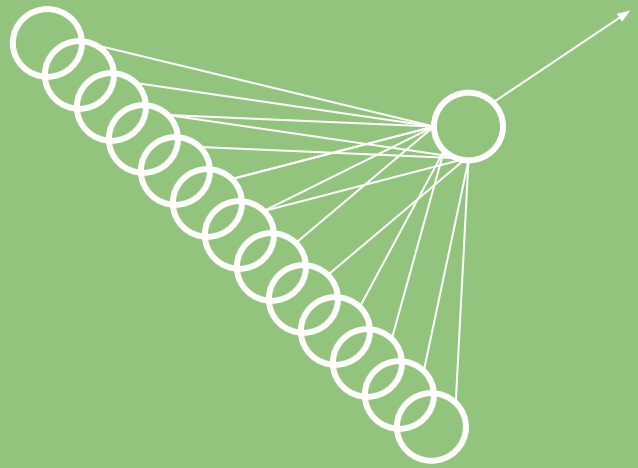
$$y = f(W, X)$$

Una red neuronal, es una función **f(w,x)** lineal o no lineal caracterizada por un conjunto de parámetros “W”. Esta función toma como entrada un vector **x** d-dimensional para generar una respuesta/salida unidimensional ‘y’ o multidimensional ‘z’. Para obtener los parámetros **w** se medirá mediante una función de costo **L** distintos aspectos como la calidad de las variables de salida de la función **f** (y o z).

Deep learning book

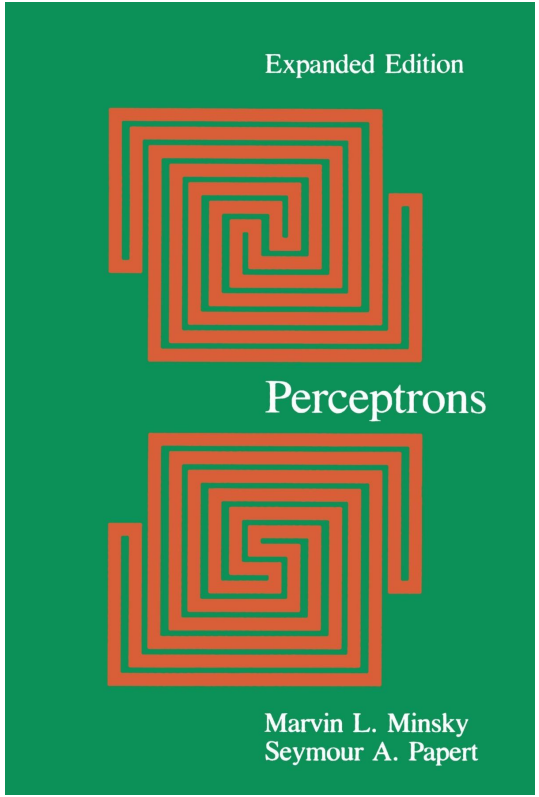


Uno de los libros más importantes en redes neuronales contemporaneas es https://www.deeplearningbook.org/



Modelo Perceptron

perceptron



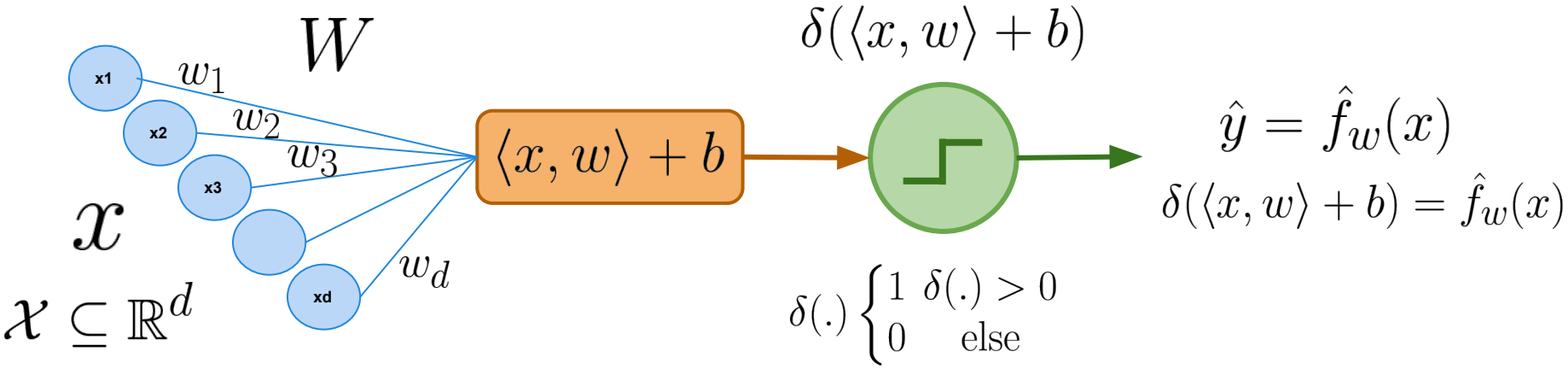
Psychological Review
Vol. 65, No. 6, 1958

THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN[1]

F. ROSENBLATT

Cornell Aeronautical Laboratory

perceptrón



Para entender las redes neuronales primero debemos estudiar el modelo **perceptrón.** Es un algoritmo de Clasificación Lineal originado en los años 50s. El modelo tiene **d** features/variable de entradas, donde cada feature de entrada estará combinada linealmente a un "peso **w**". Es decir que tenemos una combinación lineal **(x . w)**. Una vez realizada la combinación lineal se determina el valor de salida **(w*x+b) = y.** Al resultado de la combinación lineal se lo afecta por una función de activación "step" que en el caso de ser binario resultará como 0 o 1.

El objetivo es aprender los pesos **w,b** que minimizan el error de clasificación **y_hat.**

Perceptrón

- Es un modelo que trabaja “online”. Procesa una muestra a la vez. De esta manera los pesos también se actualizan gradualmente.
- Únicamente en los casos donde el modelo realiza una predicción errónea los pesos se actualizan.
- El perceptrón computa una función de costo **L(y, y_hat)** basada en la cantidad de veces que hubo una mal clasificación.
- La función de activación (‘neurona’) que utiliza es “signo o step” -> es 0 si x<0 y +1 si x>0.

Ejercicio perceptron

$$\mathbf{x}_{test} = \begin{matrix} x_1 & x_2 \end{matrix} \begin{bmatrix} 1 & 1 \\ 3 & 0.5 \\ 2 & 3 \\ 2.5 & 1.5 \\ 1.5 & 2 \\ 2.5 & 4 \\ 3.5 & 1.5 \\ 4 & 3 \end{bmatrix} \quad \mathbf{y}_{test} = \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

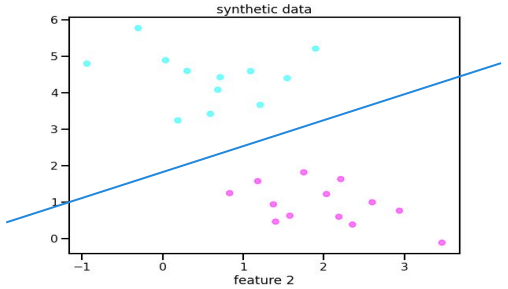
$$\delta(\langle x, w \rangle + b) = \hat{f}_w(x)$$

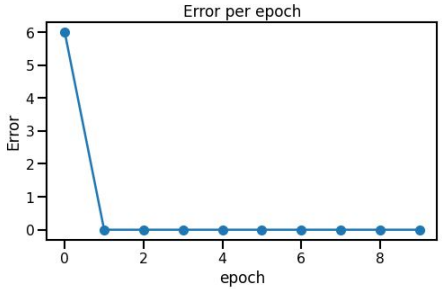
	w1	w2	b	Accuracy
$\hat{f}_1$	2	2	-9	?
$\hat{f}_2$	3.4	5	-14	?

Suponer dos modelos perceptrón f_1 y f_2 ya entrenados previamente con un dataset que no disponemos. Por lo tanto los parámetros de los modelos están disponibles. Se solicita evaluar los dos modelos utilizando las muestras de test provistas y calcular el accuracy para determinar cual de los dos generaliza mejor.

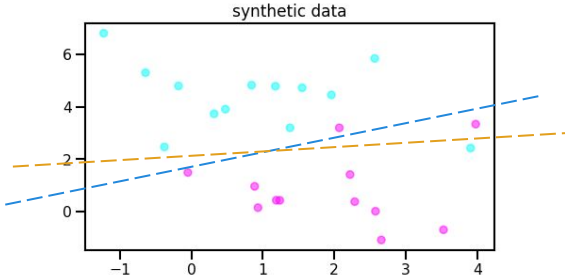
Limitaciones del perceptrón

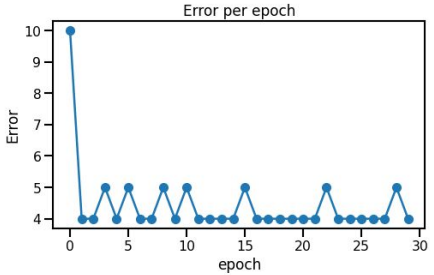
Clases separables



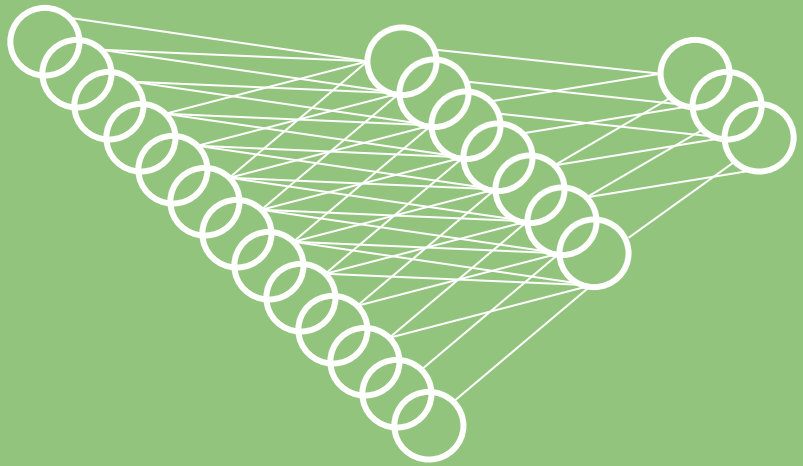


Clases linealmente no separables





Durante el entrenamiento el dataset es procesado por el perceptrón múltiples veces iterativamente donde cada iteración se la denomina “epoch”. El perceptrón únicamente actualiza los parámetros cuando el output **y_hat** es distinto a la verdadera etiqueta **y**. En clases linealmente separables el perceptron convergerá a una solución. En clases no separables el perceptrón iterara la cantidad de veces que el usuario le indique sin poder llegar a una solución definitiva.



Redes neuronales artificiales

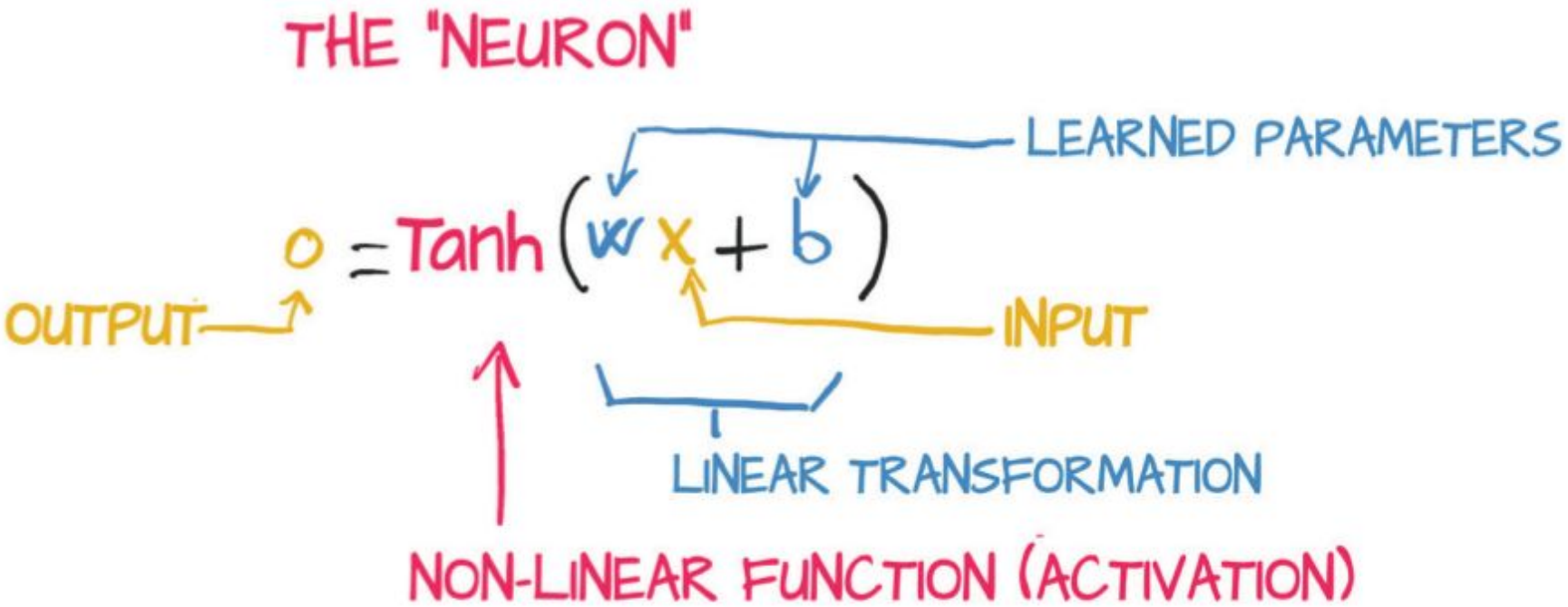
Funciones de activación

$$\mathbf{f_i} = \sigma_i(\mathbf{W_i X_{i-1}} + \mathbf{b_i})$$

La función de activación es un **nodo o neurona** que tiene múltiples entradas y define una determinada salida. Las funciones de activación pueden estar “encendidas” o “desactivadas”. Algunas populares son:

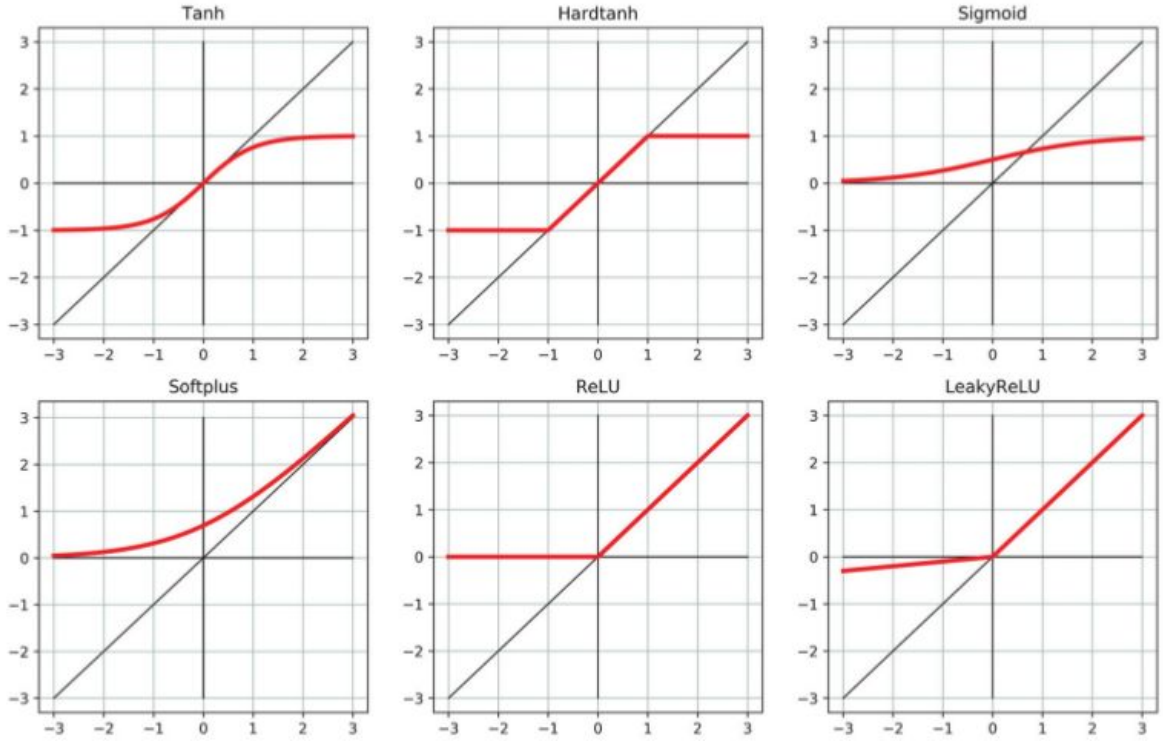
- ReLu
- Softmax
- Sigmoid
- TanH

Funciones de activación



*Deep Learning with Pytorch, 2019

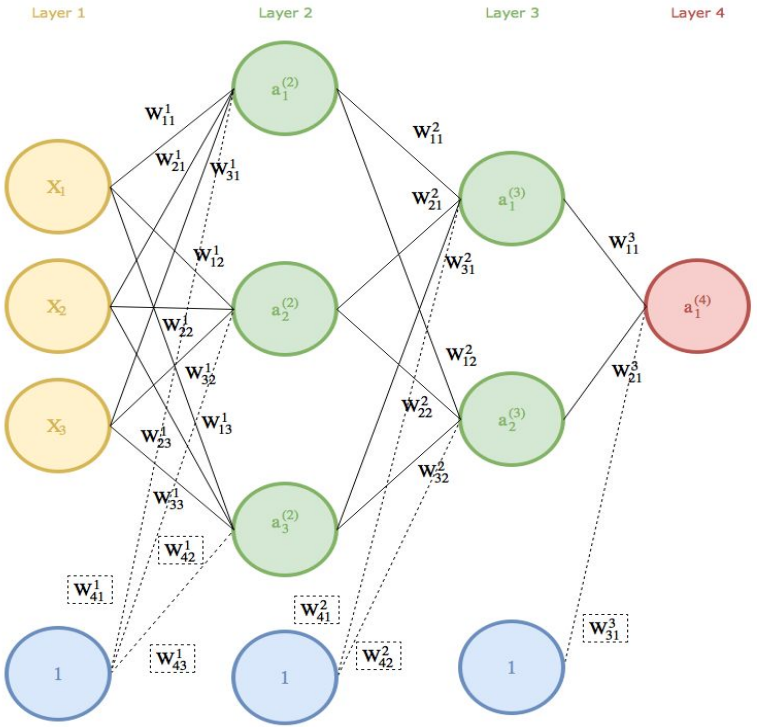
Funciones de activación (neuronas)



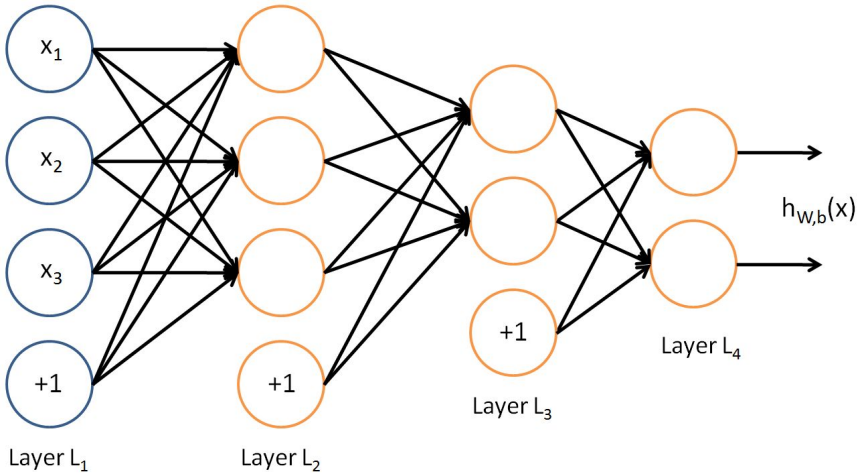
*Deep Learning with Pytorch, 2019

Multilayer Perceptrón / Feedforward Network

Si al modelo perceptrón le incorporamos capas intermedias entre la entrada y la salida obtenemos lo que comúnmente se llama “red neuronal con capas ocultas” o en inglés “neural network with hidden layers”. El arco que conecta dos neuronas se caracteriza con un peso que se aprenderá con el set de entrenamiento para minimizar el error.

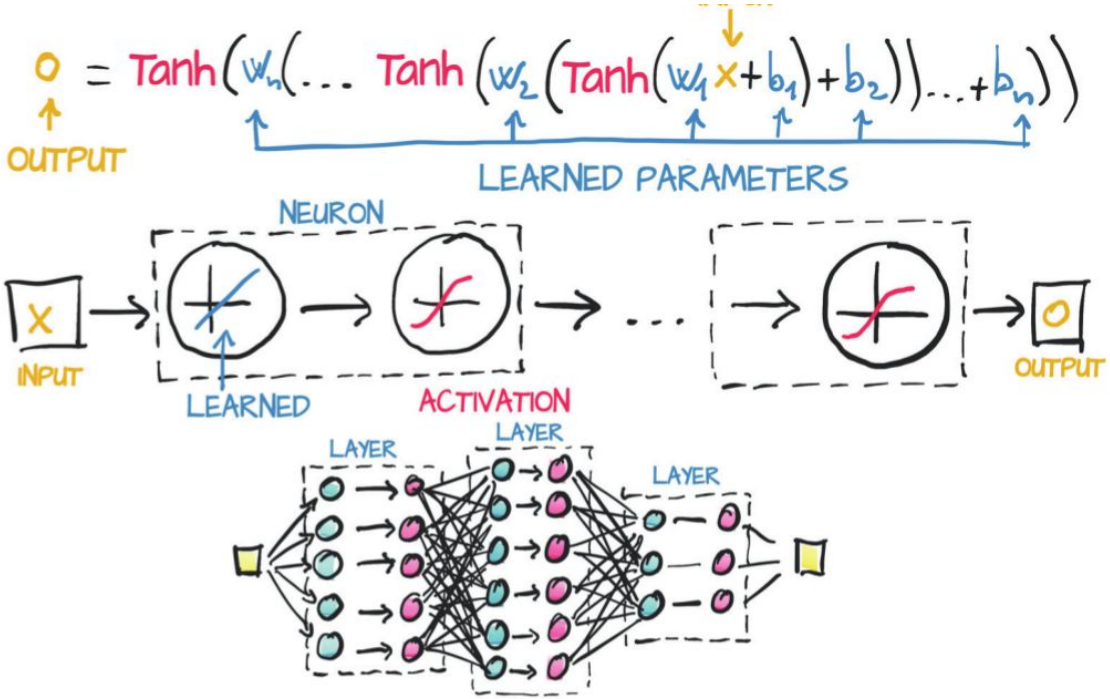


Neural Networks: arquitectura



La arquitectura multi-capa o perceptrón multi-capa permite capturar patrones mas complejos en los datos. Cada capa es una representación de los datos donde cada neurona de una capa es una feature/variable aprendida por la combinación de variables de la capa anterior. A medida que mas capas se agregan mas complejas pueden ser las variables/features aprendidas. De todos modos agregar mas capas implica agregar mas complejidad y parámetros al modelo.

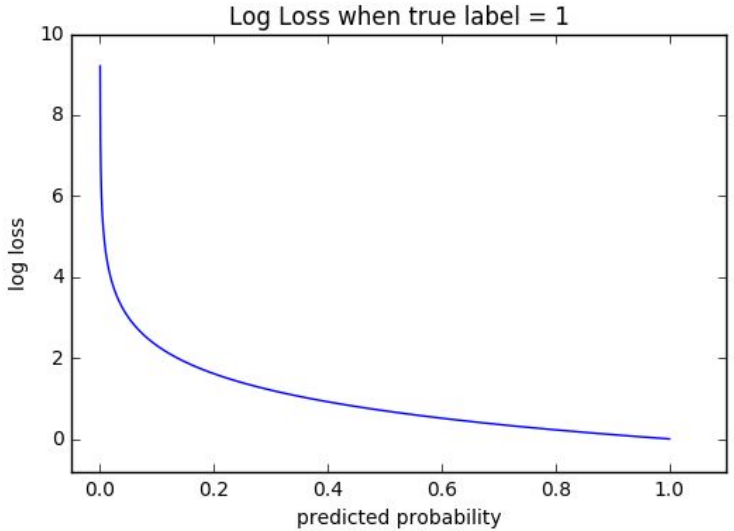
Neural Networks: arquitectura



*Deep Learning with Pytorch, 2019

Funciones de costo: Cross Entropy

Mide la performance de un modelo de clasificación cuya salida es una probabilidad de 0 a 1. El Cross Entropy Loss aumenta cuando la probabilidad predicha diverge de la etiqueta actual. Un modelo perfecto debería dar una Cross Entropy loss = 0



$$-(y\log(p)+(1-y)\log(1-p))$$

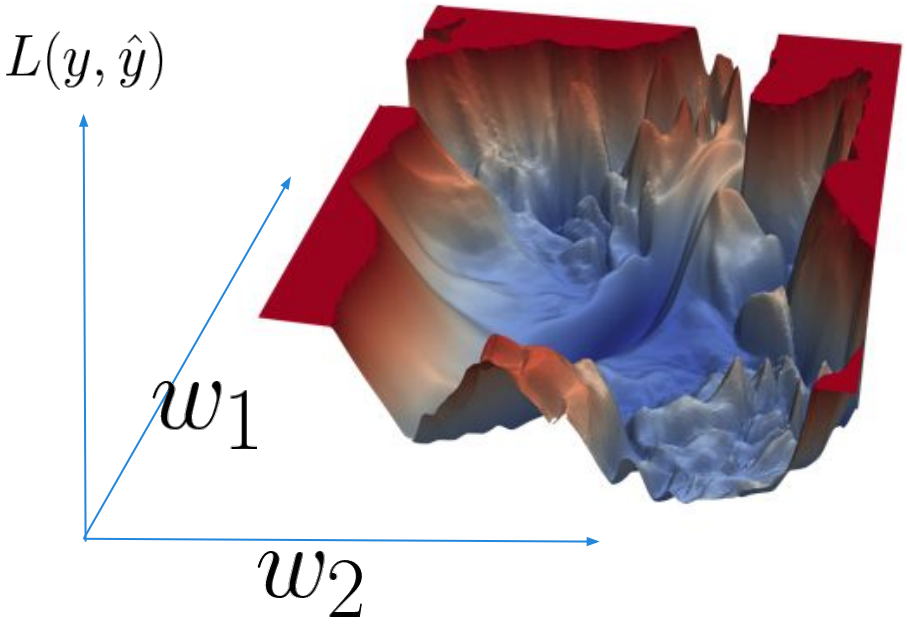
Funciones de costo: Mean Squared Error

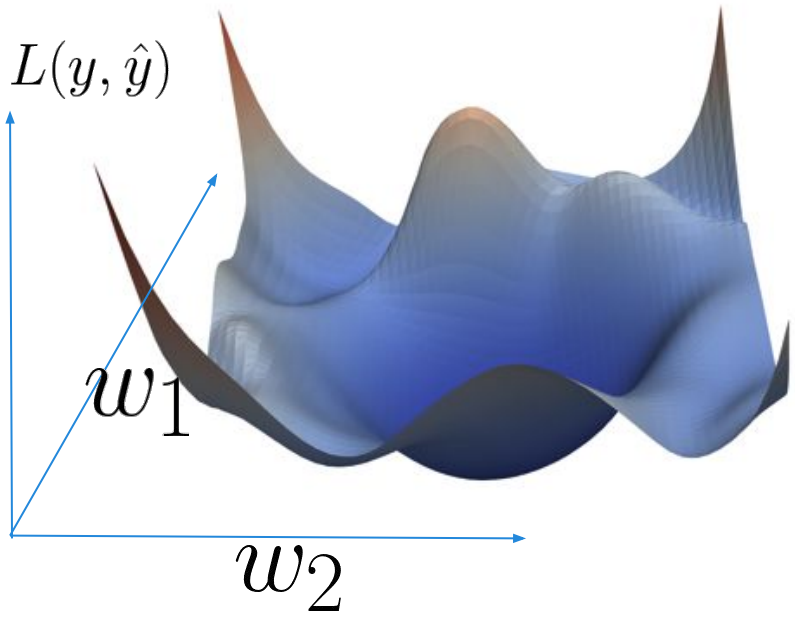
Mide la performance de un modelo de regresión cuya salida es la diferencia al cuadrado de los residuos. El MSE además de usarse en problemas de regresión puede usarse en autoencoders donde se busca reconstruir un vector d-dimensional.

$$\text{MSE} = \left\| y_i - y_j \right\|^2$$

Funciones de costo

Las funciones de costo dependen de los parámetros **w** (pesos) del modelo y del dataset en cuestión. El objetivo es encontrar un vector de parámetros que se encuentre en un minimo local aceptable de la funcion de costo.

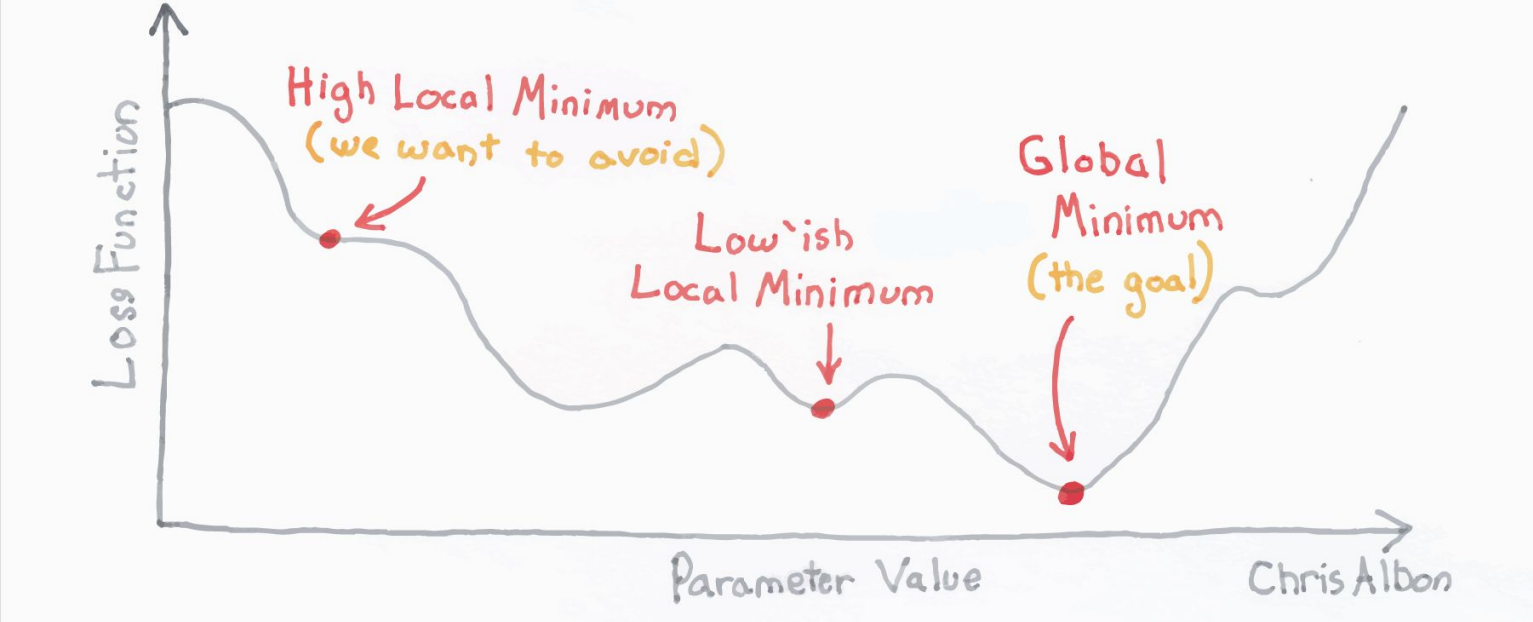




*https://www.jeremyjordan.me

Funciones de costo

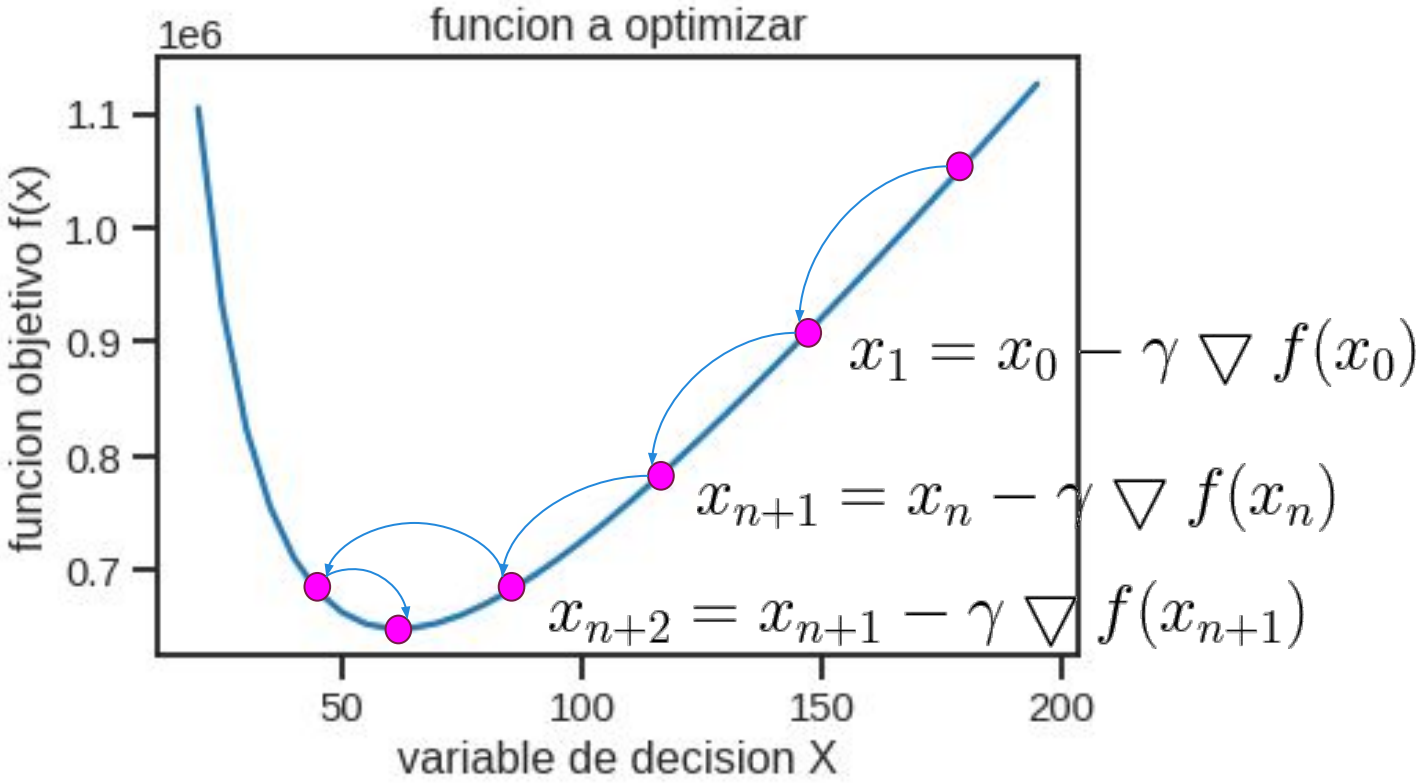
Nuestro deseo es encontrar la configuración de pesos **w** que minimiza globalmente la función de costo. A veces no conseguimos ese objetivo aunque podemos llegar a mínimos locales que permiten una performance de clasificación aceptable.



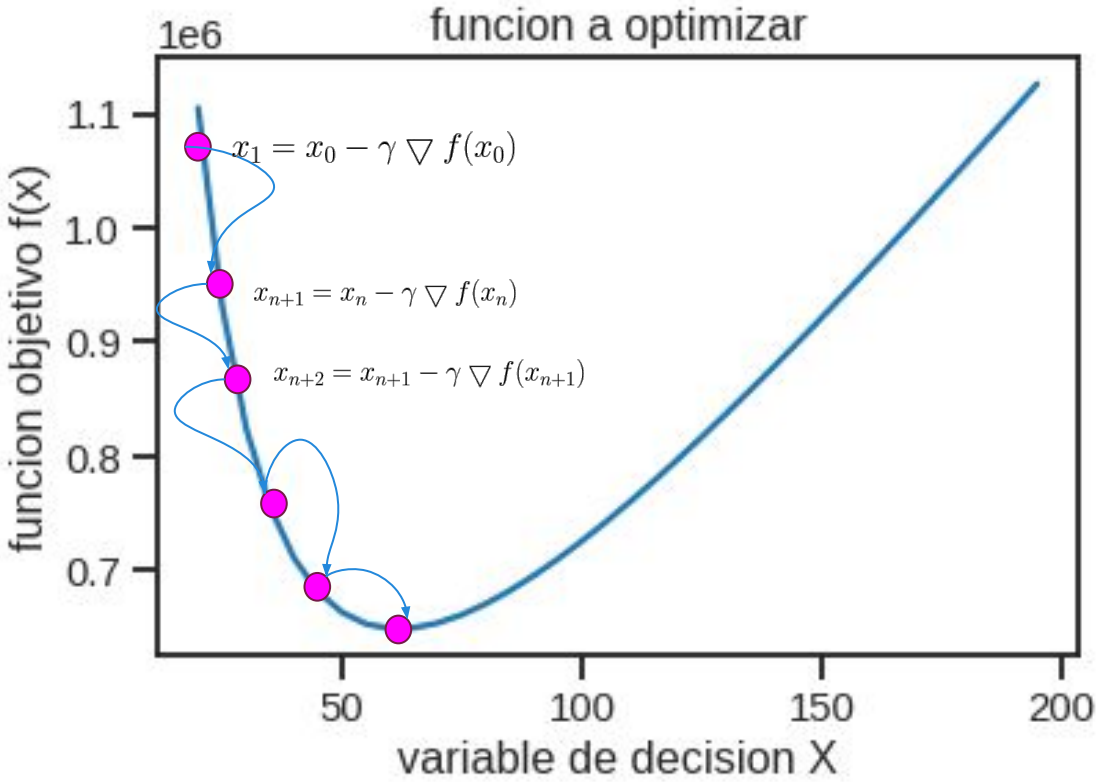
Descenso por gradiente

Metodo para resolver: optimización no lineal

Gradiente descendiente



Gradiente descendiente



Optimización por gradiente descendiente

$$x \in \mathbb{R}^d \qquad \min_x f(x) \qquad \nabla f(x_0) \qquad \gamma$$

Gradiente de F(x)

Paso de optimización

$$x_1 = x_0 - \gamma \nabla f(x_0)$$

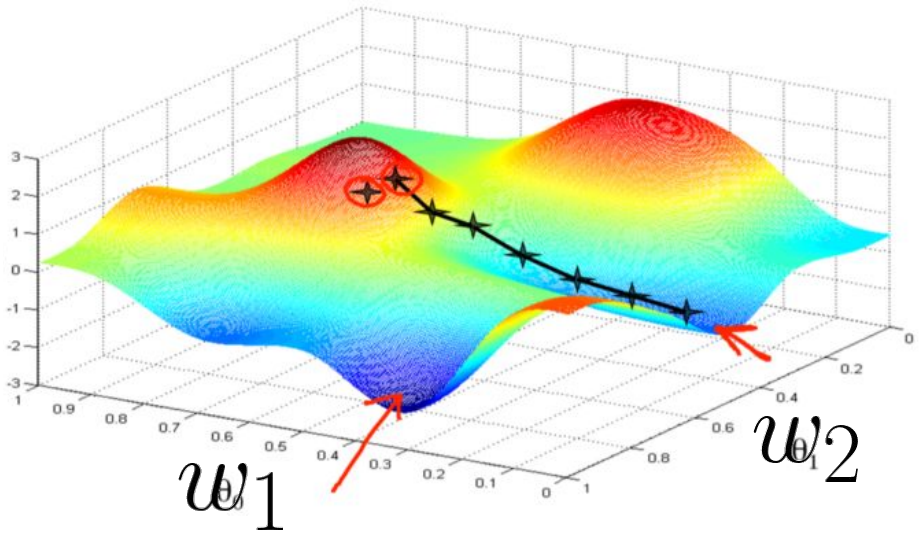
$$x_{n+1} = x_n - \gamma \nabla f(x_n)$$

$$x_{n+2} = x_{n+1} - \gamma \nabla f(x_{n+1})$$

$$f(x_n) > f(x_{n+1})$$

Supongamos que queremos optimizar la función no lineal, continua y diferenciable f(x) definida en el dominio de X. Se partirá desde la solución inicial x0 (llamado semilla) actualizando iterativamente la solución del problema en dirección al gradiente descendiente con un paso *gamma* de manera tal que la nueva solución presente un valor de la función objetivo menor.

Learning Rate y Loss Function



En este caso tenemos una función de costo determinada por dos parámetros (pesos). El Learning Rate se visualiza como los “pasos” que se dan hacia la búsqueda del mínimo. Un Learning rate bajo significa tardar mucho en llegar a un mínimo local. Un LR alto puede significar pasar “por alto” un mínimo y no converger.

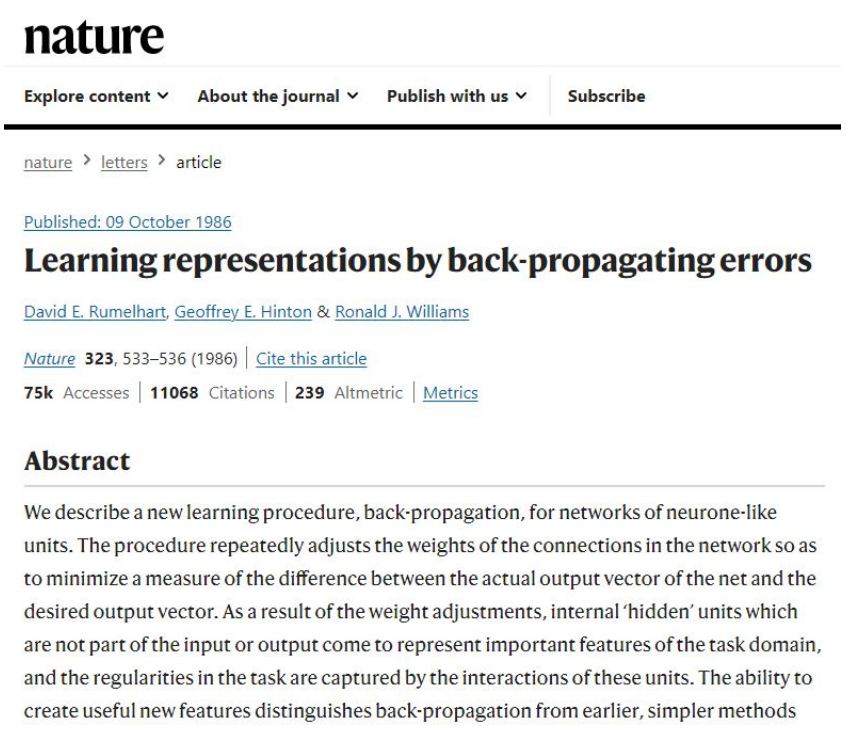
Backpropagation & Gradient Descent

Gradient Descent o Gradiente Descendiente es una técnica de optimización para explorar la función de costo y buscar el mínimo que optimizará nuestra red. El punto de mínimo costo en la función de error implica la combinación de parámetros (pesos) que minimiza el error o la mal clasificación con las muestras de train. Una vez que el gradiente descendiente encuentra un mínimo, detendrá la búsqueda y se quedará en dicha región de la función de costo. La manera en que se busca

$$w_{h+1} = w_h - \alpha \frac{\partial L}{\partial w}$$

El **Learning Rate** (alpha) es un hiperparámetro que indica cuán rápido se actualizan los pesos en búsqueda de un mínimo.

Backpropagation



nature

Explore content ▾ About the journal ▾ Publish with us ▾ Subscribe

nature › letters › article

Published: 09 October 1986

Learning representations by back-propagating errors

David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams

Nature **323**, 533–536 (1986) | Cite this article

75k Accesses | **11068** Citations | **239** Altmetric | Metrics

Abstract

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods

Backpropagation es el **algoritmo** de actualización de pesos de una red neuronal para moverse a favor del gradiente descendiente. Sucede luego que la información fluye desde la entrada input a la salida output (forward propagation) y se conoce si el modelo clasificó bien o mal a cada muestra. Mediante la **regla de la cadena** se calculan los gradientes de la función de costo en función de los parámetros **w** , es decir que se actualizan los pesos que aproximan en dirección al mínimo (local).

Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). *Deep learning* (Vol. 1, p. 2). Cambridge: MIT press. https://www.deeplearningbook.org/contents/mlp.html#pf25

Neural Networks: components

- Activation functions
- Network architecture
- Loss Function
- Weight optimization technique
- Learning rate

Neural Networks: clasificador

Softmax: Multi o binary classification

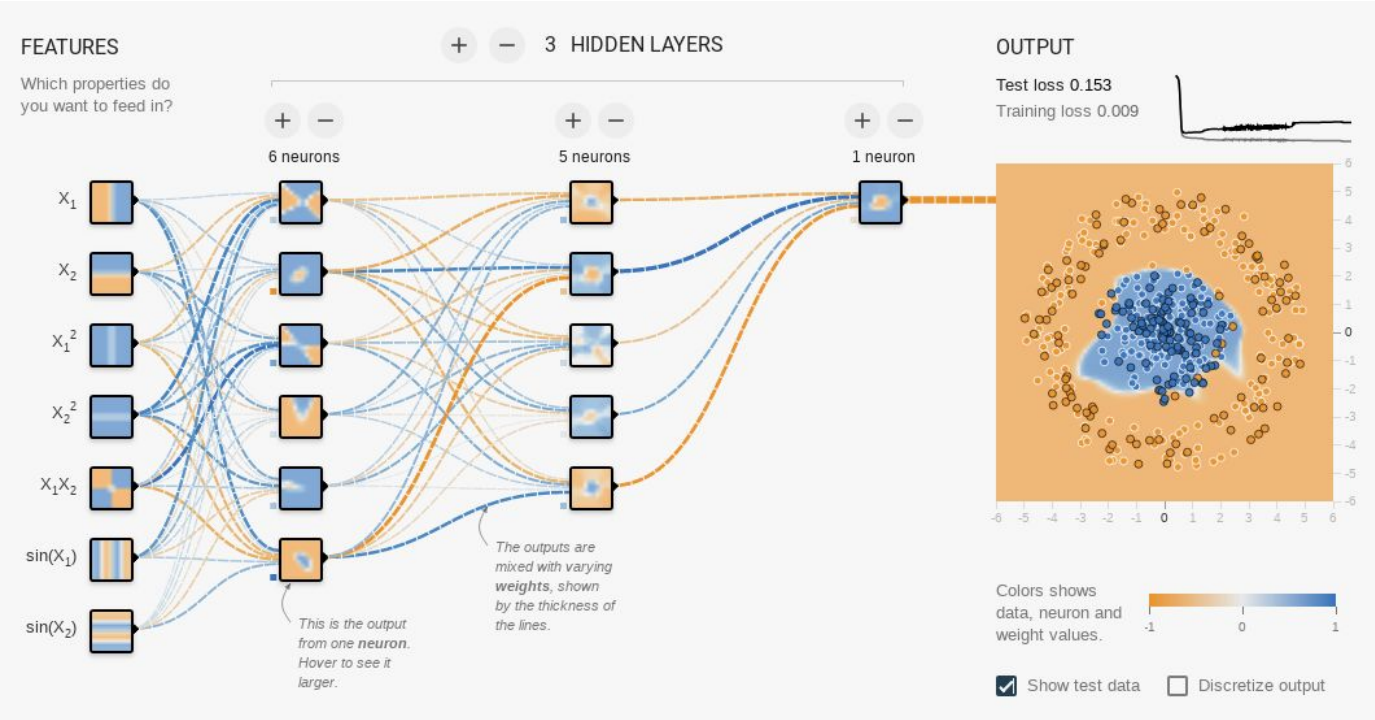
$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

Sigmoid: Binary classification

$$\sigma(\text{x}) = \frac{1}{1 + \exp(-\text{x})}$$

Al final de todas las “layers” de la red neuronal se ubica una función de activación llamada “Softmax” o ‘Sigmoid’ que arroja valores entre 0 y 1 a modo de probabilidad siendo las etiquetas 0 y 1. Los pesos de la red se aprenderán de manera tal que la salida de la softmax o sigmoid coincida lo mejor posible con las etiquetas reales de cada muestra.

Neural Networks: clasificador



*https://playground.tensorflow.org

Regularización en redes neuronales

Regularización es penalizar al modelo a la hora de aprender los parámetros internos (los pesos W). No queremos que el modelo aprenda perfectamente los datos de entrenamiento puesto que estos no son 100% representativos de la realidad. La regularización nos permite que el modelo no haga “overfitting”.

- Norma L1 sobre los pesos
- Norma L2 sobre los pesos
- Dropout en las neuronas
- Batch-Normalization

Librerías para ANNs