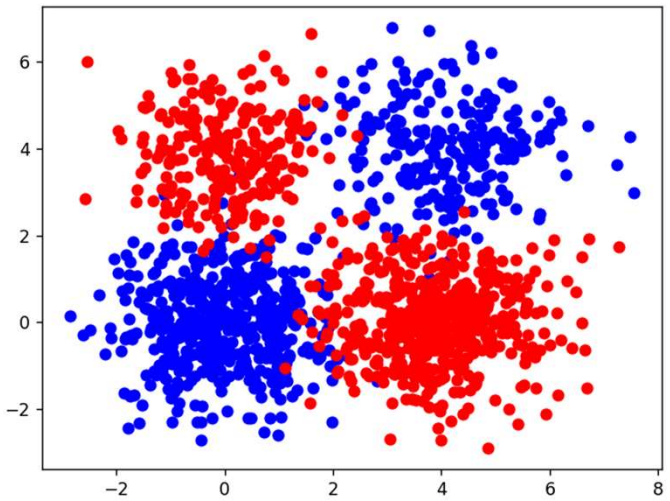
Redes neuronales

Backpropagation

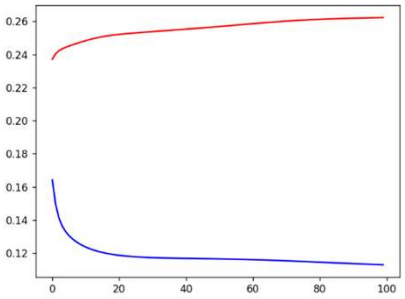
Arquitectura de red

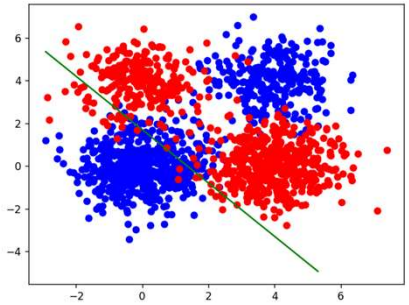
Una neurona sigmoide ha sido suficiente para realizar una tarea simple: clasificación lineal

Observemos este dataset:



Al entrenar una
Neurona sigmoide



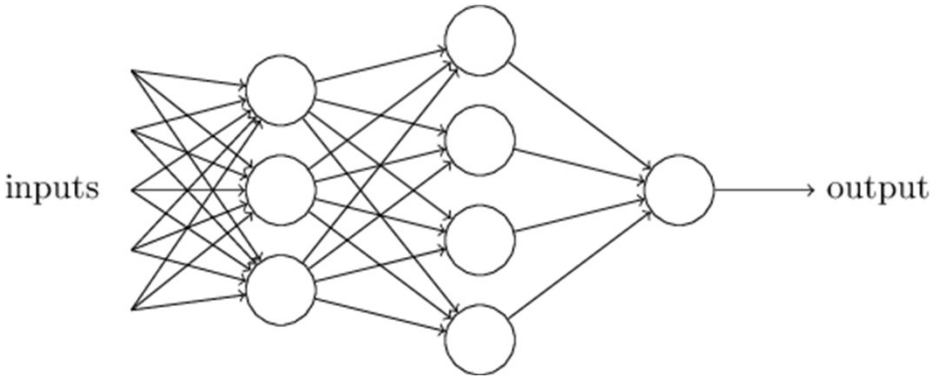


El modelo no es lo suficientemente complejo para aprender la data

Perceptron Multicapa

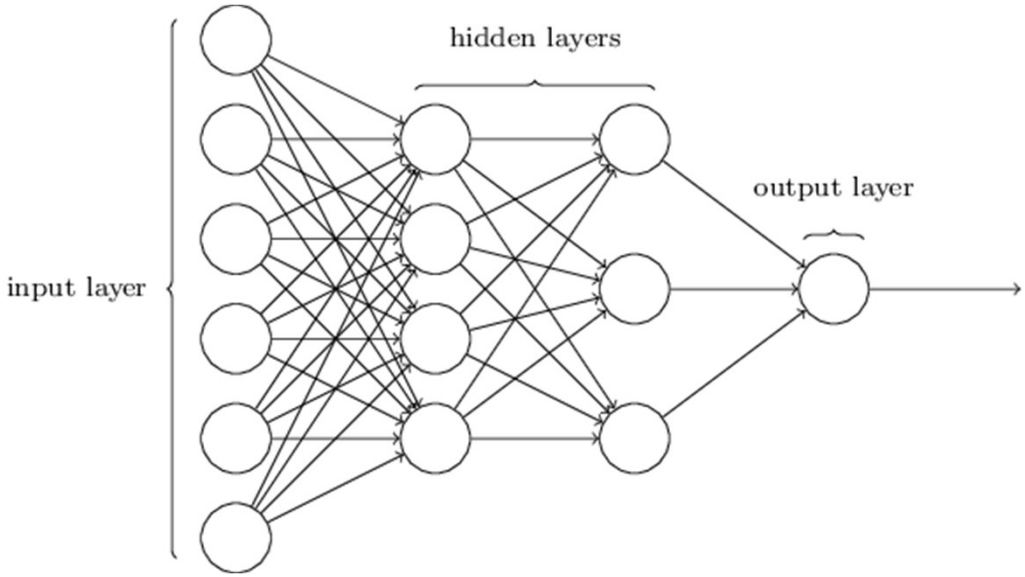
Un perceptrón implementa una regla de decisión muy simple. Es imposible solucionar un problema real con una sola neurona.

Para hacer reglas de decisión más complejas, podemos implementar conexiones complejas de perceptrones



Perceptrones pueden ser organizados en capas: capas de entrada (reciben los datos), capa de salida (computa la salida), y capas ocultas (para tomar decisiones internas)
Este modelo es comúnmente conocido como Multi-layer perceptron o MLP.

Arquitectura de Redes Neuronales



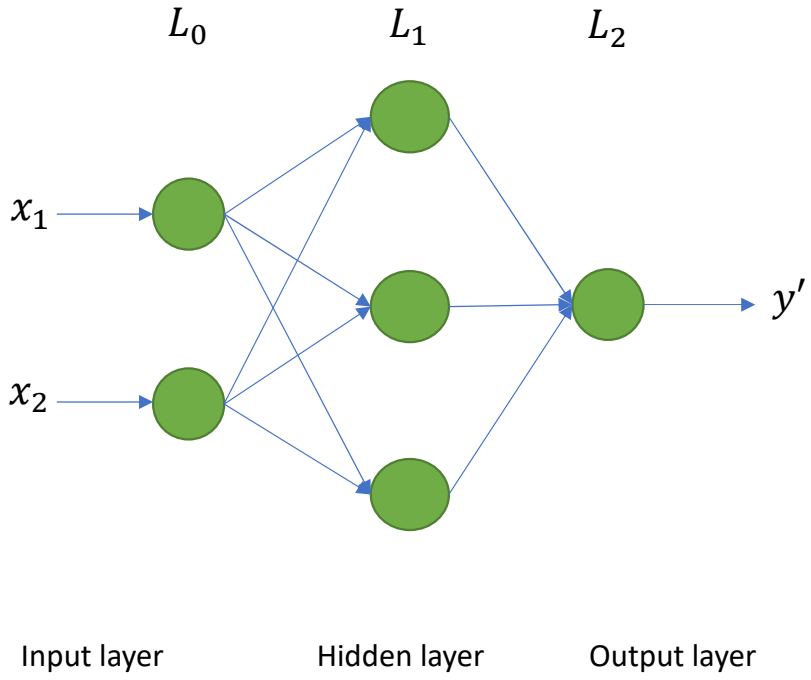
Multi-layer perceptron

Feedforward neural networks

- Número de neuronas de entrada depende de la cantidad de data entrante
- Número de neuronas de salida depende del problema también
  - Para clasificación binaria, una neurona es suficiente
  - Para problemas multi-clase, one-hot encoding funciona
- Número de capas ocultas y número de neuronas en capas ocultas es un misterio

Arquitectura de red

Diseñamos una red de neuronas sigmoide: secuencia de capas con neuronas



Neuronas de entrada son representadas pero no realizan computación

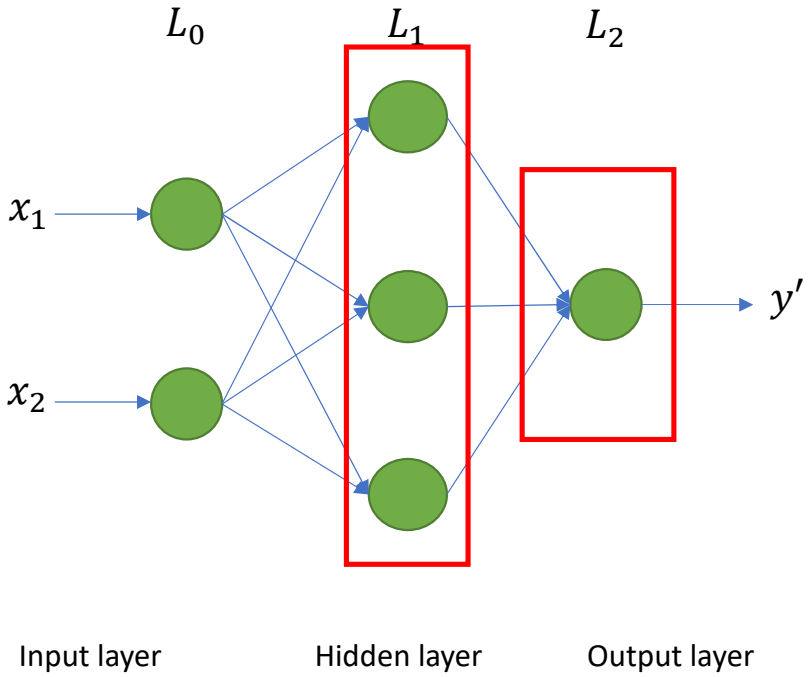
Toda flecha en esta figura representa un peso (parámetro)

Por conveniencia, empaquetamos los pesos de la primera capa oculta en una matriz W^1 de dimensión 2×3. Pesos en la capa de salida son empaquetados en una matriz W^2 de dimensión 3×1.

Biases son organizados en vectores: b^1 de dimensión 3 y b^2 de dimensión 1.

Arquitectura de red

Diseñamos una red de neuronas sigmoide: secuencia de capas que contienen neuronas



Computación de la red

Sea $x = [x_1, x_2]$ un vector fila, computamos

$$X_1 = \sigma(x \cdot W^1 + b^1)$$

$$y' = \sigma(X_1 \cdot W^2 + b^2)$$

Un vector de dimensión 3
Suma con b es un broadcast

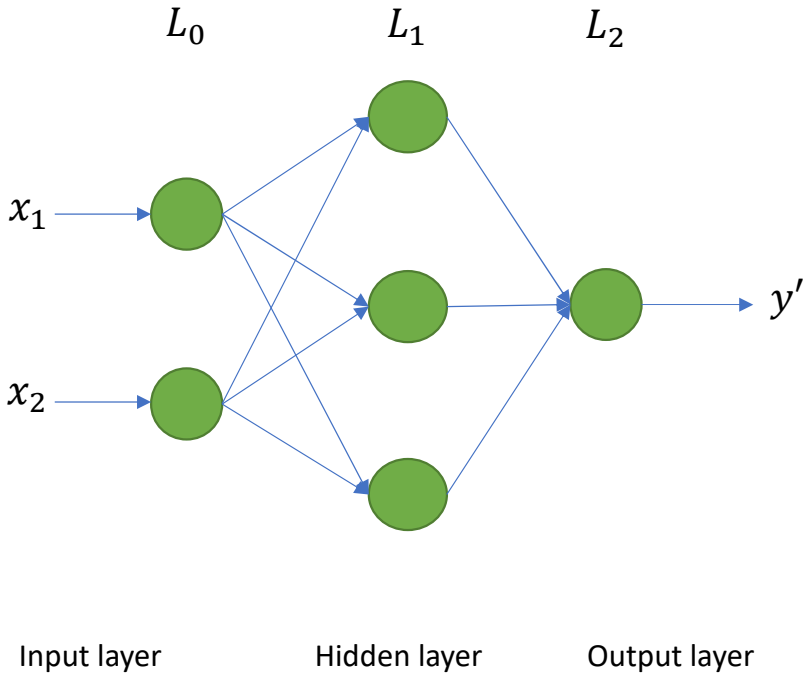
Por favor, chequear dimensiones

En general, la salida de una capa es

$$X_i = \sigma(X_{i-1} \cdot W^i + b^i)$$

Arquitectura de red

Diseñamos una red de neuronas sigmoide: secuencia de capas que contienen neuronas



Y el algoritmo de entrenamiento?

A primera vista, la computación de gradientes parece más complicada ahora. Sin embargo, veremos una estrategia.

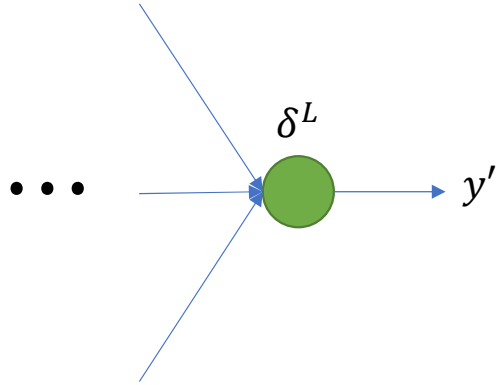
Definición: dada la j-ésima neurona en la capa l, el error se define como

$$\delta_j^l = \frac{\partial C}{\partial z_j^l}$$

Donde, C es la función de costo y z es el resultado de la ecuación lineal en la neurona

Arquitectura de red

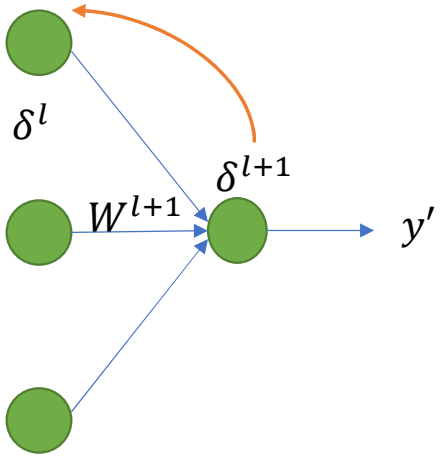
El error en la capa de salida



$$\delta^L = \frac{\partial C}{\partial y'}\sigma'(z^L)$$

$$= (y' - y) \cdot \sigma'(z^L)$$

El error de una capa en términos del error de la siguiente capa



$$\delta^l = (\delta^{l+1}(W^{l+1})^T) \odot \sigma'(z^l)$$

Error va hacia atrás a través de los pesos

Arquitectura de red

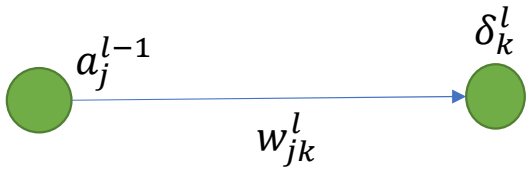
Gradiente de biases

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

El gradiente en el bias, es de hecho,el error

El gradiente de los pesos

$$\frac{\partial C}{\partial w_{jk}^l} = a_j^{l-1}\delta_k^l$$



Backpropagation

Algoritmo GD
Input: Conjunto de muestras $\{(x_1, x_2, y)\}$, learning rate η, número de épocas E

Para cada muestra de entrenamiento x: la activación de entrada es a^1

- **Feedforward:** Para cada capa $l = 2, 3, \cdots, L$ computar

$$z^l = a^{l-1}W^l + b^l$$
$$a^l = \sigma(z^l)$$

- **Error de salida:** Computar

$$\delta^L = \nabla C \cdot \sigma'(z^l)$$

- **Retropropagar el error**: Para cada capa $l = L-1, L-2, \cdots$, computar

$$\delta^l = (\delta^{l+1}(W^{l+1})^T) \odot \sigma'(z^l)$$

- **Computar gradiente**

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \qquad \frac{\partial C}{\partial w_{jk}^l} = a_j^{l-1}\delta_k^l$$

Resumen

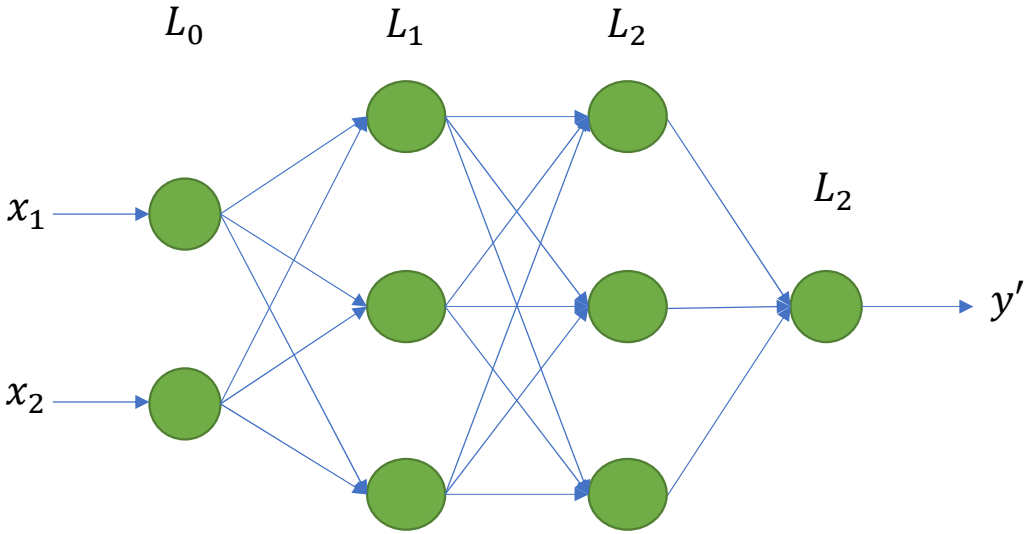
- Arquitecturas multicapa
- Reglas de Backpropagation
- Algoritmo backward para actualizar parámetros

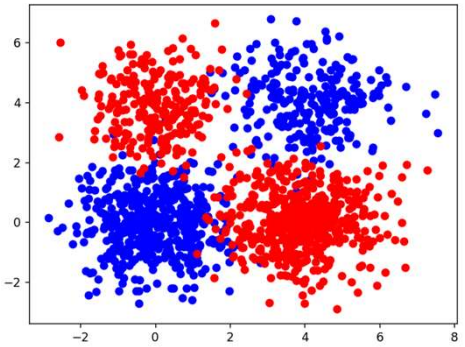
Redes neuronales

Funciones de activación

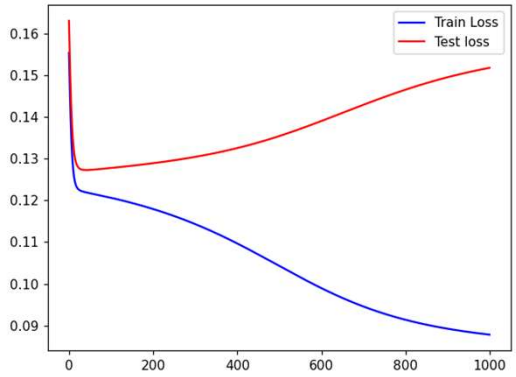
De nuevo el problema

Modelamos la red con la siguiente arquitectura:





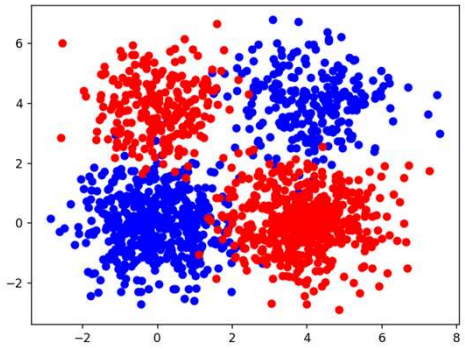
Red entrenada con SGD, batch size de 20, 1000 épocas, $\eta = 0.01$

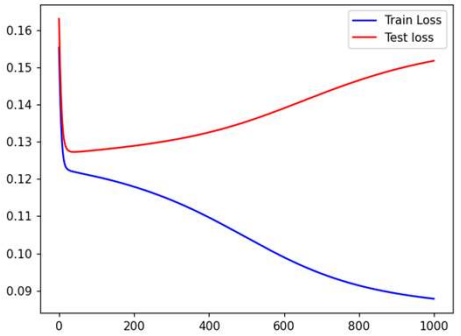


Sin embargo, la red no aprende. Optimización sufre.

De nuevo al problema

Existen muchos razones para este comportamiento:

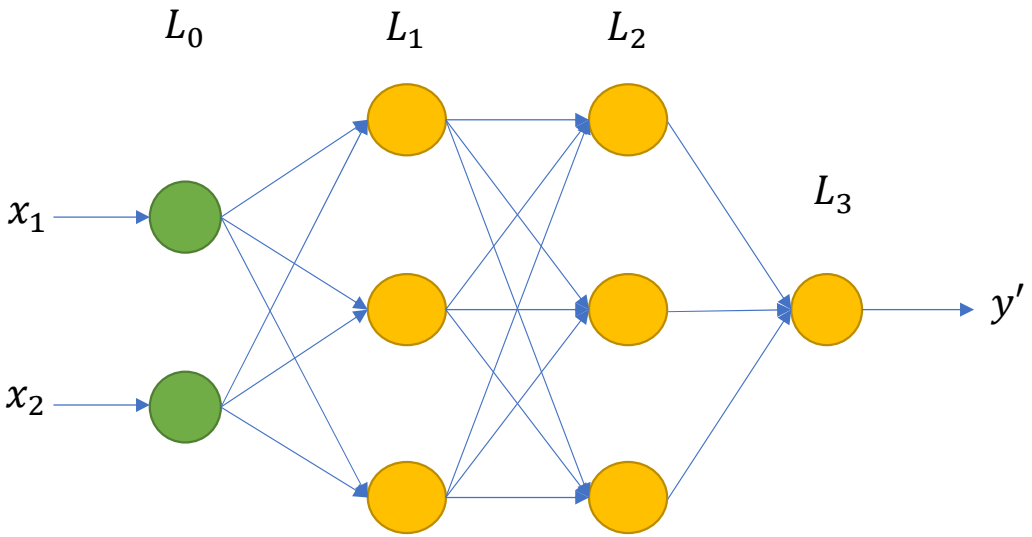




- Overfitting: aprendizaje es (casi) bueno para la data de entrenamiento pero mala para el test
- Modelo no es suficiente: el problema no es tan difícil
- Malos hiperparámetros: learning rate o número de épocas

Una razón posible más: la función de activación y su rol en el aprendizaje

Análisis de función de activación



Neuronas sigmoide

De las ecuaciones del error, notamos algo

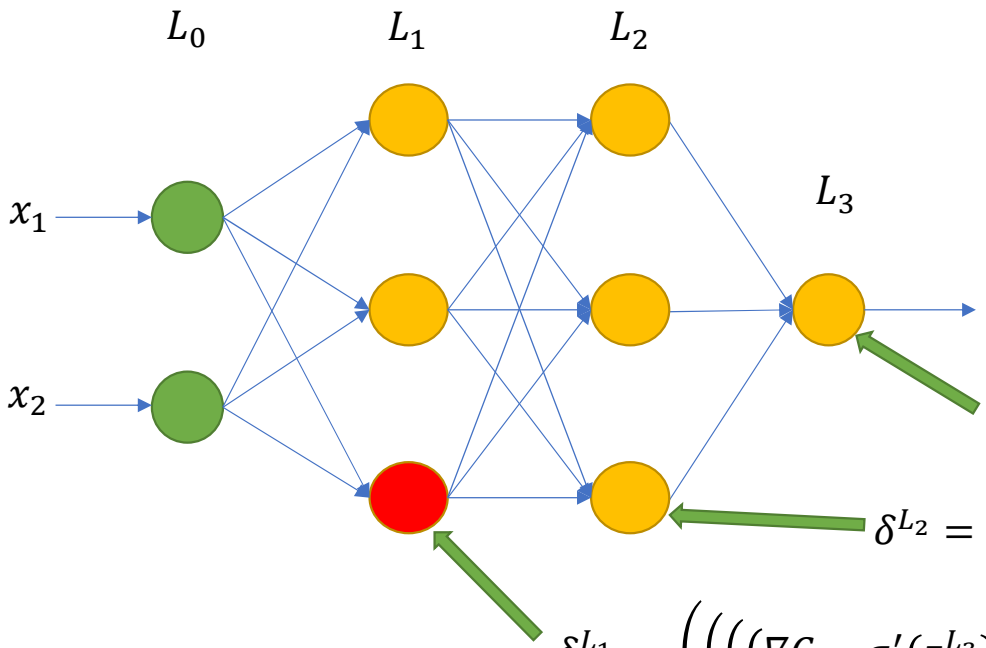
$$\delta^{L_3} = \nabla C \cdot \sigma'(z^{L_3})$$ ← Output layer

$$\delta^{l} = (\delta^{l+1}(W^{l+1})^{T}) \odot \sigma'(z^{l})$$ ← Hidden layer

Todo error depende de la derivada de la función de activación
La derivada es retropropagada para computar errores en cada neurona
Error en las primeras capas depende del error en capas posteriores

Análisis de la función de activación

Analicemos el error en una cierta neurona:



$$\delta^{L_3} = \nabla C \cdot \sigma'(z^{L_3})$$

$$\delta^{L_2} = \left(\left(\nabla C \cdot \sigma'(z^{L_3})\right)(W^3)^T\right) \odot \sigma'(z^{L_2})$$

$$\delta^{L_1} = \left(\left(\left(\left(\nabla C \cdot \sigma'(z^{L_3})\right)(W^3)^T\right) \odot \sigma'(z^{L_2})\right)(W^2)^T\right) \odot \sigma'(z^{L_1})$$

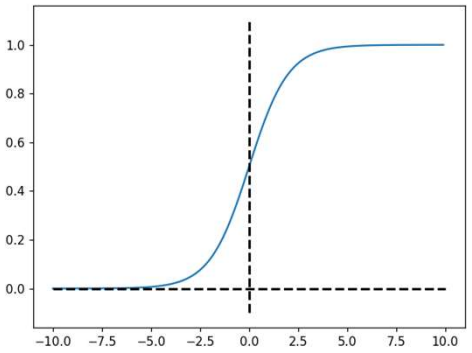
Si queremos mirar el efecto de la función de activación en el error de esta neurona, el error puede ser expresado como

$$\rightarrow \quad X \cdot \sigma'(z^{L_3}) \cdot \sigma'(z^{L_2}) \cdot \sigma'(z^{L_1})$$

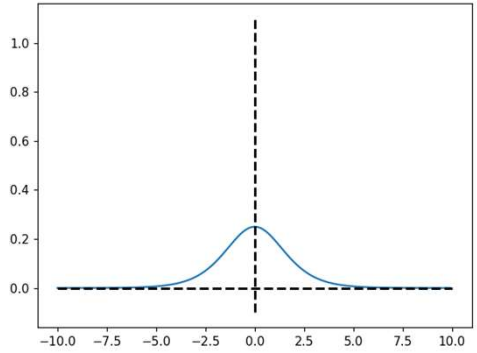
Análisis de la función de activación

Analicemos el error en una cierta neurona:

Función sigmoide



Derivada del sigmoide



El máximo valor de la derivada ocurre en $z = 0$ con valor 0.25

Por lo tanto, se cumple que

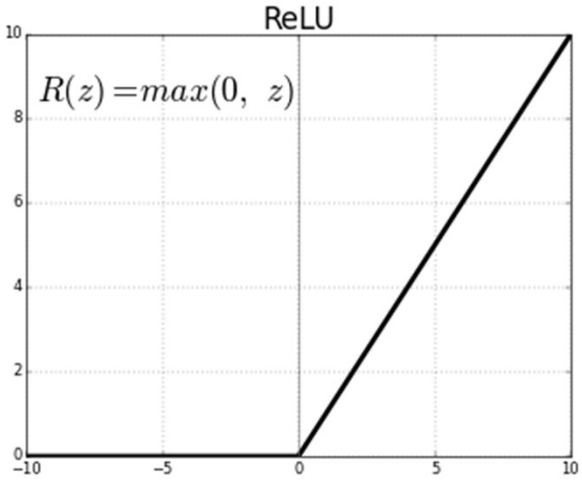
$$X \cdot \sigma'(z^{L_3}) \cdot \sigma'(z^{L_2}) \cdot \sigma'(z^{L_1}) \leq \frac{1}{64} X$$

El gradiente en las primeras capas se degrada a cero, porque la derivada del sigmoide decrementa el error en un factor de $^1/_4$ en cada capa.

La búsqueda de una función de activación

La función sigmoide no es útil en arquitecturas profundas, porque los gradientes se desvanecen
El éxito de los métodos de Deep learning es gracias a funciones de activación más efectivas

La función ReLU

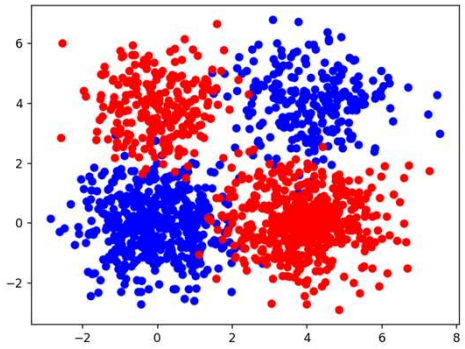


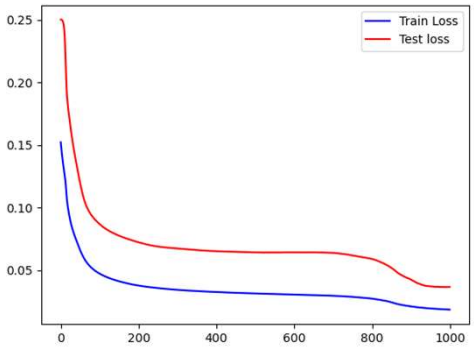
- Función más simple
- Valores negativos se truncan a cero
- El gradiente es muy simple:

$$\frac{\partial R}{\partial z} = \begin{cases} 1 & z > 0 \\ 0 & z \leq 0 \end{cases}$$

Volvemos al problema

Usando ReLU en las capas ocultas y sigmoide en la capa final, obtenemos





- Accuracy de entrenamiento: 94%
- Accuracy de test: 90.4%

Resumen

- Gradientes desvanecidos con función sigmoide
- Rectified Linear Units (ReLU)